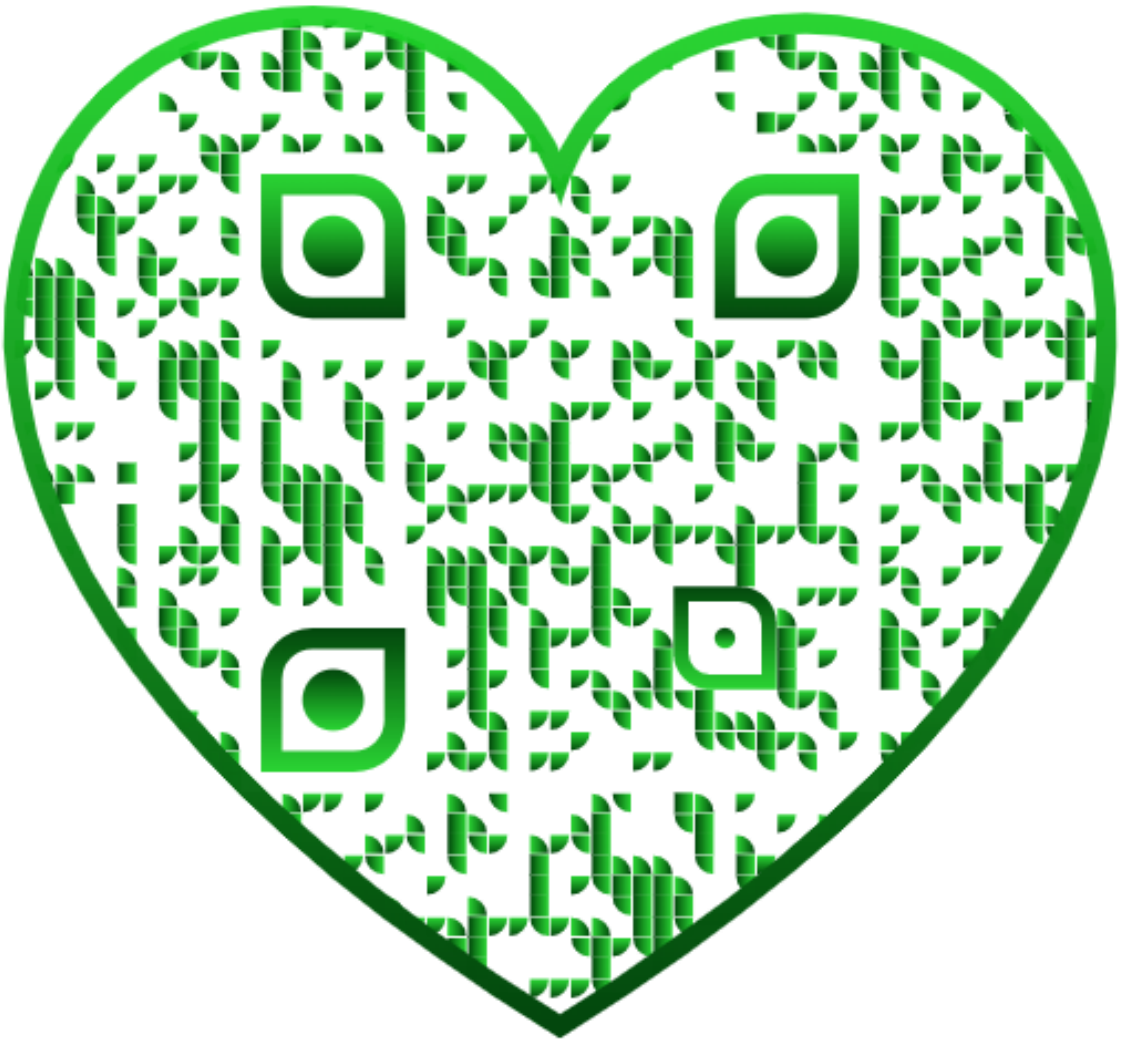
Michal Zalewski

Menaces sur le réseau

Sécurité informatique : guide pratique des attaques passives et indirectes

PEARSON



Menaces sur le réseau

Guide pratique des attaques passives et indirectes

Michal Zalewski

PEARSON

CampusPress a apporté le plus grand soin à la réalisation de ce livre afin de vous fournir une information complète et fiable. Cependant, CampusPress n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Les exemples ou les programmes présents dans cet ouvrage sont fournis pour illustrer les descriptions théoriques. Ils ne sont en aucun cas destinés à une utilisation commerciale ou professionnelle.

CampusPress ne pourra en aucun cas être tenu pour responsable des préjudices ou dommages de quelque nature que ce soit pouvant résulter de l'utilisation de ces exemples ou programmes.

Tous les noms de produits ou marques cités dans ce livre sont des marques déposées par leurs propriétaires respectifs.

Publié par CampusPress
47 bis, rue des Vinaigriers
75010 PARIS
Tél. : 01 72 74 90 00

Mise en pages : TyPAO
Collaboration éditoriale :
Marie-France Claerebout

ISBN : 978-2-7440-4031-3

Copyright© 2009
Pearson Education France
Tous droits réservés

CampusPress est une marque
de Pearson Education France

Titre original :
Silence on the wire. A Field Guide to Passive Reconnaissance and Indirect Attacks

Traduit de l'américain par Philippe Beaudran

Relecture technique : Paolo Pinto (Sysdream)

ISBN original : 1-59327-046-1
Copyright © 2005 by Michal Zalewski
All rights reserved

No Starch Press

www.nostarch.com

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Aucune représentation ou reproduction, même partielle, autre que celles prévues à l'article L. 122-5 2° et 3° a) du code de la propriété intellectuelle ne peut être faite sans l'autorisation expresse de Pearson Education France ou, le cas échéant, sans le respect des modalités prévues à l'article L. 122-10 dudit code.

Pour Maja

À propos de l'auteur

Michal Zalewski est un chercheur autodidacte sur la sécurité de l'information qui a travaillé sur des sujets allant de la conception du matériel et des systèmes d'exploitation à la gestion des réseaux. Il a découvert de nombreux bogues, est un membre actif de Bugtraq depuis le milieu des années 1990 et a écrit des utilitaires de sécurité populaires comme p0f, un utilitaire de fingerprinting passif du système d'exploitation. Il a également publié un certain nombre d'études sur la sécurité qui ont été acclamées. Michal a travaillé comme expert pour la sécurité dans plusieurs entreprises de renom, dont deux géants des télécommunications, à la fois dans son pays natal, la Pologne, et aux États-Unis. En plus d'être un fervent chercheur et un programmeur occasionnel, Michal s'intéresse également à l'intelligence artificielle, aux mathématiques appliquées, à l'électronique et à la photographie.

Remerciements

Nous remercions vivement Paolo Pinto (Sysdream) pour sa précieuse collaboration et son soutien technique, sans lesquels l'édition française de cet ouvrage n'aurait pu voir le jour.

Table des matières

À propos de l'auteur V
Remerciements V

Avant-propos XV

Introduction 1
Quelques mots de l'auteur 1
À propos de cet ouvrage 2

Partie I – La source

1. L'écoute du clavier 7
Le besoin de hasard 8
Générer automatiquement des nombres aléatoires 10
La sécurité des générateurs de nombres pseudo-aléatoires 11
Entropie entrée-sortie : votre souris vous parle 12
Interruptions : un exemple pratique 13
Fonctions de résumé 15
De l'importance d'être pédant 17
L'entropie ne doit pas être gâchée 18
Attaques : les implications d'un changement soudain de paradigme 19
Examen plus détaillé des motifs dans le temps en entrée 20
Tactiques de défense immédiates 24
Les générateurs de nombres aléatoires matériels : une meilleure solution ? 24
Matière à réflexion 26
Les attaques à distance fondées sur le temps 26
Exploiter les diagnostics du système 27
Reproduction de l'imprévisible 27

2. Des efforts supplémentaires ne sont jamais inutiles 29
L'héritage de Boole 30

Vers l'opérateur universel 30
La loi de De Morgan en pratique 31
La commodité est une nécessité 32
Englober la complexité 33
Vers le monde matériel 34
Un ordinateur sans électricité 35
Une conception d'ordinateur légèrement plus classique 36
Portes logiques 36
Des opérateurs logiques aux calculs 37
Du sablier électronique à l'ordinateur 40
La machine de Turing et les suites complexes d'instructions 43
Utilisation pratique, enfin 45
Le Graal : l'ordinateur programmable 45
Des améliorations par simplification 46
Le partage des tâches 47
Étapes d'exécution 48
Le moins de mémoire possible 50
Plus d'actions à la fois : le pipelining 51
Le gros problème des pipelines 52
Implications : de subtiles différences 53
Utiliser les motifs dans le temps pour reconstruire les données 54
Bit par bit 55
En pratique 56
Optimisation early-out 57
Code fonctionnel, faites-le vous-même 59
Prévention 61
Matière à réflexion 62

3. Les dix têtes de l'hydre 63
TEMPEST : l'espionnage des émissions TV 64
Les limitations de la confidentialité 66
Établir la provenance des données 67
Divulgation malencontreuse : *_~1q'@@... et le mot de passe est… 68

4. Travailler pour le bien de tous 71

Partie II – Un endroit sûr

5. Les Blinkenlights 79
L'art de transmettre des données 80
De votre courrier électronique à des bruits intenses… aller et retour 82
De nos jours 88
Parfois, un modem est juste un modem 89
Les collisions sous contrôle 90
Les coulisses : la soupe de câble et comment la gérer 93
Les blinkenlights et les communications 95
Les conséquences des diodes esthétiques 96
Construire son propre dispositif d'espionnage… 97
…et l'utiliser avec un ordinateur 99
Empêcher que les DEL ne divulguent des données (et pourquoi cela ne fonctionne pas) 103
Matière à réflexion 106

6. Échos du passé 109
Construire la tour de Babel 110
Le modèle OSI 111
La phrase manquante 112
Matière à réflexion 115

7. La sécurité dans les réseaux commutés 117
Un peu de théorie 118
La résolution de l'adresse et la commutation 118
Les réseaux virtuels et la gestion du trafic 120
Attaques sur l'architecture 123
Le CAM et l'interception du trafic 123
Autres exemples d'attaques : DTP, STP, trunks 124
Prévention des attaques 124
Matière à réflexion 125

8. L'enfer, c'est les autres 127
Les indicateurs logiques et leur utilisation inhabituelle 129
Montrez-moi ce que vous tapez et je vous dirai qui vous êtes 130
Les bits inattendus : des données personnelles disséminées partout 131
Les failles du wi-fi 132

Partie III – Dans la jungle

9. Un accent étranger 137
Le langage d'Internet 138
Routage naïf 139
Le routage dans le monde réel 140
L'espace d'adressage 141
Des empreintes digitales sur l'enveloppe 143
Internet Protocol 143
Version du protocole 144
Le champ IHL 145
Le champ Service (8 bits) 145
La longueur totale (16 bits) 145
L'adresse IP source 146
L'adresse IP de destination 146
L'identifiant du protocole de la quatrième couche 146
Le TTL 146
Les paramètres Flags et Offset 147
Numéro d'identification 149
Somme de contrôle 150
Au-delà du protocole IP 150
Le protocole UDP 151
Introduction à l'adressage des ports 152
Résumé de l'en-tête UDP 153
Les paquets TCP 153
Flags de contrôle : la poignée de main TCP 154
Autres paramètres de l'en-tête TCP 158
Options TCP 159
Les paquets ICMP (Internet Control Message) 162
Le fingerprinting passif 164
Examiner les paquets IP : les origines 164
Time to Live initial (couche IP) 165
Le flag DF (couche IP) 165
Le numéro IP ID (couche IP) 166
Type de service (couche IP) 166
Les champs Nonzero et Must Be Zero (couches IP et TCP) 167
Port source (couche TCP) 168
Taille de fenêtre (couche TCP) 168
Pointeur d'urgence et valeurs du numéro d'acquittement (couche TCP) 169

L'ordre des options (couche TCP) 169
Décalage de fenêtre (couche TCP, option) 170
Maximum Segment Size (TCP Layer, option) 170
Données Timestamp (couche TCP, option) 170
Autres types de fingerprinting passif 171
Le fingerprinting passif en pratique 171
Les applications de fingerprinting passif 174
Collecter des données statistiques et des incidents de connexion 174
Optimisation du contenu 175
Élaboration d'une politique 175
La sécurité du pauvre 175
Test de sécurité et découverte du réseau 176
Profiling du consommateur et invasion de la vie privée 176
Espionnage et reconnaissance furtive 176
Protection contre le fingerprinting 177
Matière à réflexion : le défaut fatal de la fragmentation IP 178
Fragmentation TCP 180

10. Stratégies avancées pour compter les moutons 183
Les avantages et les implications du fingerprinting passif classique 184
Un bref historique des numéros de séquence 186
Obtenir plus d'informations des numéros de séquence 188
Coordonnées temporisées : images des séquences dans le temps 189
Une galerie de jolies images de la pile TCP/IP 194
Attaques utilisant les attracteurs 199
Retour au fingerprinting du système 202
ISNProber, la mise en pratique de la théorie 203
Empêcher l'analyse passive 204
Matière à réflexion 204

11. La reconnaissance des anomalies 207
Les bases des pare-feu de paquets 208
Filtrage et fragmentation sans états 209
Filtrage sans états et perte de synchronisation du trafic 210
Les filtres de paquets à états 212
Réécriture de paquet et NAT 213
Lost in Translation 214
Les conséquences du masquerading 215
La taille des segments 216

Suivi à états et réponses inattendues 218
Fiabilité ou performances : la controverse sur le bit DF 219
Cas d'échec du Path MTU Discovery 219
La lutte contre PMTUD et ses retombées 221
Matière à réflexion 222

12. Fuite des données de la pile 225
Le serveur de Kristjan 225
Des découvertes surprenantes 226
Révélation : la reproduction du phénomène 228
Matière à réflexion 229

13. Fumée et miroirs 231
L'usurpation d'adresse IP : le scan de port avancé 232
L'arbre qui cache la forêt 232
L'idle scan 233
Se défendre contre l'idle scan 236
Matière à réflexion 236

14. L'identification du client : vos papiers, s'il vous plaît ! 237
Camouflage 238
Définition du problème 239
Vers une solution 240
Une (très) brève histoire du Web 240
Notions élémentaires sur le protocole de transfert hypertexte 242
Améliorer HTTP 244
La réduction de la latence : du bricolage 244
La mise en cache du contenu 246
La gestion des sessions : les cookies 249
Le mélange des cookies et du cache 250
Empêcher l'attaque utilisant les cookies en cache 251
La découverte des trahisons 252
Exemple simple d'analyse comportementale 253
Donner un sens aux graphiques 255
Au-delà du moteur... 256
… et au-delà de l'identification 257
Prévention 259
Matière à réflexion 259

15. Les avantages d'être une victime 261

Connaître les mesures de l'attaquant 262

Se protéger : observer les observations 266

Matière à réflexion 267

Partie IV – Vision d'ensemble

16. Le calcul parasitaire, ou comment l'union fait la force 271

L'utilisation des processeurs distants 272

Considérations pratiques 275

Les débuts du stockage parasitaire 277

Rendre possible le stockage parasitaire 279

Applications, considérations sociales, et défense 287

Matière à réflexion 288

17. La topologie du réseau 289

Capturer l'instant 290

Utiliser les données de topologie pour identifier l'origine du trafic 292

La triangulation du réseau à l'aide des données de la topologie maillée 294

L'analyse de la saturation du réseau 295

Matière à réflexion 298

18. En regardant le vide 299

Les tactiques d'observation directe 300

L'analyse des retombées du trafic de l'attaque 303

Détecter les données malformées ou mal dirigées 305

Matière à réflexion 307

Épilogue 309

Notes bibliographiques 311

Index 317

Avant-propos

Que faut-il pour écrire un livre sur la sécurité informatique ? Ou, plutôt, que faut-il pour écrire un roman sur l'informatique moderne ?

Il faut un auteur jeune mais très expérimenté qui possède des talents dans de nombreux domaines, dans de nombreux aspects de l'informatique, des mathématiques et de l'électronique (et peut-être un intérêt pour la robotique) mais qui s'intéresse également à d'autres sujets apparemment sans rapport (y compris, disons-le, la photographie érotique). Enfin, bien sûr, il doit avoir l'envie d'écrire et le talent pour le faire.

> *Il était une fois dans une forêt sombre et vierge des arbres qui, grâce à la magie des arbres (les cellules du cerveau), donnèrent naissance à un bit d'information. Dès sa naissance, il partit naviguer sur une rivière aux flots tumultueux qui se jetait dans une mer immense (Internet) pour fonder un nouveau foyer, mourir ou peut-être prendre place dans un musée. Et ainsi commence notre récit. Que notre bit soit bon ou mauvais, la rivière le conduit dans un premier temps dans un resplendissant château blanc (bien que beaucoup le considèrent encore comme une boîte noire). Il franchit l'entrée et s'approche de la réception pour signaler son arrivée. S'il n'était pas si naïf, il aurait pu constater un groupe de bits patibulaires qui observent l'arrivée des bits à distance, en notant la fréquence à laquelle ils se présentent et quittent la réception. De toute façon, il n'a d'autre choix que de s'enregistrer.*

> *Une fois reposé, notre héros est invité à se joindre à un groupe de ses congénères. Ensemble, ils s'entassent sur un radeau pneumatique déjà fort usé et dont le fond est jonché de déchets (mais s'agit-il bien de déchets ?) sans doute laissés là par le groupe qui les a précédés. En respectant les feux de circulation et en se faufilant dans les embouteillages, nos bits parviennent finalement à gagner un refuge sûr où ils accostent. Seront-ils repérés depuis les châteaux et les phares alentour ? Quelqu'un surveille-t-il les feux de circulation depuis leur départ ? Quelqu'un allume-t-il les lumières sur le quai où ils débarquent et les prend en photo ? Les méchants bits usurpent-ils leur identité et les précèdent-ils ? Nos bits ne le savent pas. Et, donc, ils embarquent sur un autre bateau et naviguent vers la mer... Le voyage de notre confrérie de bits se poursuit, et de nombreux dangers les guettent.*

Non, le livre de Michal ne cache pas de détails techniques sous l'apparence d'un conte de fées, comme je viens de le faire. Tout en restant divertissant, il énonce directement tous les faits et donne rapidement des réponses à la plupart des problèmes énoncés au début de chaque chapitre.

Bien que *Menaces sur le réseau* soit unique à de nombreux égards, deux caractéristiques lui permettent de se démarquer des autres ouvrages : tout d'abord, il fournit des informations détaillées sur la quasi-totalité des étapes essentielles du traitement des données qui permettent aujourd'hui "l'interconnexion", depuis la pression sur une touche du clavier jusqu'au résultat que cette action entraîne. Il définit ensuite les problèmes de sécurité si souvent négligés et peu étudiés qui sont inhérents à chaque étape de la mise en réseau et à l'ensemble du processus. Les questions de sécurité abordées montrent l'art de la recherche des failles aussi bien du point de vue de l'attaquant que du défenseur et encouragent le lecteur à poursuivre ses propres recherches.

Manifestement, un ouvrage sur la sécurité informatique ne peut pas être exhaustif. Dans *Menaces sur le réseau*, Michal a choisi délibérément de ne pas aborder les attaques et les failles très connues et pourtant très dangereuses sur lesquelles se concentre la majorité de la communauté qui se consacre aux problèmes de sécurité de l'information. Il vous parlera des attaques fondées sur le temps d'opération des touches du clavier mais ne vous rappellera pas que les "chevaux de Troie" qui permettent de contrôler à distance un ordinateur sont actuellement à la fois plus fréquents et plus faciles à utiliser qu'aucune de ces attaques.

Pourquoi parler de l'écoute du clavier et laisser de côté les chevaux de Troie ? Parce que les attaques fondées sur le temps sont largement sous-estimées et mal comprises, même par les professionnels de la sécurité, tandis que les chevaux de Troie sont bien connus et représentent une menace évidente. La vulnérabilité aux attaques fondées sur le temps est une propriété

inhérente à la conception de nombreux composants, tandis qu'implanter un cheval de Troie exige soit un bogue du logiciel soit qu'un utilisateur final fasse une erreur.

De même, et à quelques exceptions près, vous ne trouverez pas la moindre mention dans cet ouvrage des bogues logiciels les plus courants ni même des bogues des classes génériques des logiciels comme le "dépassement de tampon". Si les failles de sécurité informatiques les plus communes ne vous sont pas familières et que vous désiriez en savoir plus sur ces sujets, vous devrez pour suivre ce livre vous plonger dans des lectures moins passionnantes sur Internet et dans d'autres ouvrages, en particulier dans les documents qui abordent le système d'exploitation spécifique que vous utilisez.

Mais vous vous demandez peut-être pourquoi étudier le silence, puisque le silence n'est rien. Oui, dans un sens mais, de ce point de vue, le zéro aussi n'est rien. Or zéro est aussi un nombre, un concept sans lequel nous ne pouvons pas vraiment comprendre le monde.

Appréciez autant que possible le silence.

Alexander Peslyak

Fondateur et directeur technique de la société Openwall

mieux connu sous le pseudo

Solar Designer

Chef de projet Openwall

Janvier 2005

Introduction

Quelques mots de l'auteur

On pourrait croire que je suis un *geek* informatique depuis l'enfance, mais ma rencontre avec les problèmes de sécurité des réseaux s'est produite par accident. J'ai toujours aimé expérimenter, explorer de nouvelles idées et relever des défis en théorie bien définis mais pas si simples en pratique ; des problèmes qui exigent pour les résoudre d'adopter une approche novatrice et créative – même si j'échoue finalement à trouver une solution. Quand j'étais jeune, j'ai passé la plus grande partie de mon temps à effectuer des tentatives parfois risquées et souvent stupides en chimie, mathématiques, électronique et finalement en informatique plutôt que faire du vélo dans mon quartier toute la journée (j'exagère probablement un peu, mais ma mère semblait toujours être inquiète).

Peu de temps après ma première rencontre avec Internet (au milieu des années 1990, peut-être huit ans après avoir codé mon premier programme "Hello world" sur une machine à 8 bits que j'adorais), j'ai reçu une demande inhabituelle par e-mail : un mailing de masse qui, c'est difficile à croire, me demandait (ainsi qu'à plusieurs milliers d'autres personnes) de rejoindre une équipe de hackers à chapeau noir. Ce message ne m'a pas poussé à entrer dans ce milieu (peut-être en raison de mon fort instinct de conservation, ce que d'autres considèrent comme de la lâcheté), mais il m'a en quelque sorte incité à explorer le domaine de la sécurité informatique plus en détail. Ayant fait beaucoup de programmation en amateur, j'étais fasciné par l'idée de considérer le

code avec un autre point de vue et d'essayer de trouver un moyen pour qu'un algorithme fasse quelque chose de plus que ce qu'il était censé faire. Internet me semblait parfait pour les défis dont je rêvais – un système important et complexe reposant sur un seul principe : on ne peut vraiment avoir confiance en personne. Et c'est ainsi que tout a commencé.

Je n'ai pas la culture que vous pourriez attendre d'un spécialiste en sécurité informatique, une profession qui est en train de devenir monnaie courante aujourd'hui. Je n'ai jamais reçu aucune éducation formelle en science informatique et je ne suis pas bardé de diplômes. La sécurité a toujours constitué une de mes principales passions (et maintenant ma profession). Je ne suis pas le stéréotype du geek informatique et je prends de temps à autre de la distance par rapport à mon travail, voire m'éloigne complètement de tout ordinateur.

Que cela soit un mal ou un bien, tout cela a influencé la forme et le contenu de ce livre. Je donne ma vision de la sécurité informatique, d'une manière différente de l'enseignement traditionnel. Pour moi, la sécurité ne représente pas un seul problème à résoudre ni un processus unique à suivre. Il s'agit non pas de maîtriser un domaine spécifique mais de voir l'ensemble de l'écosystème et de comprendre chaque composant.

À propos de cet ouvrage

Même dans la faible lueur dispensée par nos écrans, nous ne sommes encore que des êtres humains. Nous avons appris à faire confiance aux autres, et nous ne voulons pas être trop paranoïaques. Nous avons besoin de trouver un compromis raisonnable entre sécurité et productivité pour vivre confortablement.

Néanmoins, Internet est différent du monde réel. Se conformer aux règles n'apporte aucun avantage à l'ensemble et commettre des méfaits ne nous laisse généralement que peu de remords. Nous ne pouvons tout simplement pas faire confiance au système et il n'existe pas de règle unique qui puisse être appliquée à tous les problèmes. Instinctivement, nous traçons une ligne de démarcation entre "nous" et "eux" afin de créer une île depuis laquelle nous regardons les navires pirates naviguer à l'horizon. Bientôt, des problèmes de sécurité commencent à apparaître, des anomalies localisées qui peuvent être facilement identifiées, analysées et résolues. De ce point de vue, les intentions des agresseurs semblent être claires et, si nous sommes vigilants, nous pouvons les voir approcher et les arrêter.

Pourtant, le monde virtuel est différent : sécurité ne signifie pas absence de bogues ; la sécurité ne consiste pas à rester hors de portée des attaques. Chaque processus ou presque qui implique le traitement ou la transmission d'information a sur la sécurité des conséquences qui deviennent visibles à l'instant où nous regardons au-delà de l'objectif

que le processus essaie d'atteindre. Comprendre la sécurité est l'art de franchir cette ligne de démarcation et d'adopter un point de vue différent.

Ce livre n'est pas conventionnel, du moins je l'espère. Il ne s'agit ni d'un recueil de problèmes ni d'un guide pour sécuriser vos systèmes. Il commence par suivre l'histoire des informations, depuis le moment où vos mains touchent le clavier jusqu'à leur arrivée à l'autre extrémité du réseau. Il traite de considérations technologiques et de leurs conséquences sur la sécurité, en mettant l'accent sur des problèmes qui ne peuvent pas être qualifiés de bogues, car il n'y a pas d'attaquant, pas de faille à analyser ni à résoudre ou parce que ces attaques sont indétectables (elles ne peuvent du moins pas être distinguées de l'activité normale). L'objectif de ce livre est de démontrer que le seul moyen de comprendre Internet est d'avoir le courage de dépasser les spécifications ou de les lire entre les lignes.

Comme le sous-titre l'indique, ce livre met l'accent sur la confidentialité et les problèmes de sécurité inhérents aux communications et à l'informatique de tous les jours. Certains de ces problèmes ont des conséquences importantes, tandis que d'autres sont simplement intéressants et stimulants. Aucun n'aura d'impact dévastateur immédiat sur l'environnement ou ne détruira les données du disque dur. Cet ouvrage s'adresse aux professionnels et aux amateurs chevronnés des technologies de l'information qui veulent réfléchir et qui désirent en apprendre davantage sur les conséquences peu évidentes des décisions prises lors de la conception. Il se destine à ceux qui veulent apprendre à utiliser ces subtilités pour prendre le contrôle de leur environnement et acquérir un avantage sur le monde extérieur.

Ce livre est divisé en quatre sections. Les trois premières parties abordent les différentes étapes de la transmission des données et des technologies qui sont déployées pour cela. La dernière section porte sur le réseau dans son ensemble. Chaque chapitre traite des technologies utilisées à chaque étape du traitement des données, examine leurs implications en matière de sécurité, montre leurs effets secondaires, indique si possible des moyens d'aborder ces problèmes et comment explorer le sujet plus en détail. Dans la mesure du possible, j'essaie d'éviter les graphiques, tableaux et autres spécifications (mais vous trouverez de nombreuses références en bas de page). Comme vous pouvez facilement trouver beaucoup de documentation de référence en ligne, mon objectif est avant tout de rendre ce livre agréable à lire.

Nous commençons ?

Partie I

La source

Où il est question des problèmes qui surgissent bien avant que les informations ne soient envoyées sur le réseau.

1

L'écoute du clavier

Où l'on apprend que les touches du clavier peuvent être surveillées à distance.

Dès le moment où vous appuyez sur une touche du clavier, l'information que vous envoyez commence un long périple dans le monde virtuel. Quelques microsecondes avant que les paquets ne transitent par les fibres optiques ou par satellite, chaque morceau d'information commence un tortueux voyage dans un labyrinthe de circuits. Bien avant que le système d'exploitation et les programmes qu'il exécute ne la reçoivent, de nombreux mécanismes subtils et de bas niveau entament un processus qui intéresse toutes sortes de hackers et qui s'est révélé avoir une grande signification pour les responsables de la sécurité. Le chemin qui mène à l'utilisateur est pavé de nombreuses surprises.

Ce chapitre est consacré à ces phases initiales d'envoi des données et aux possibilités qu'ont les autres utilisateurs (bienveillants ou non) d'en savoir beaucoup sur ce que vous faites devant votre terminal.

Un exemple de divulgation d'information à travers la façon dont l'ordinateur traite les données que vous saisissez renvoie à un sujet qui semble pourtant totalement sans rapport à première vue : la difficulté de produire des chiffres aléatoires sur une machine qui se comporte de façon totalement prévisible. Il est en effet difficile de trouver un rapport aussi ténu mais, pourtant, ce problème existe bien et peut permettre à un

observateur sournois de déduire énormément de choses à partir de l'activité d'un utilisateur, qu'il s'agisse de ses mots de passe ou des e-mails confidentiels qu'il saisit.

Le besoin de hasard

Les ordinateurs sont totalement déterministes. Ils traitent les données en fonction d'un ensemble de règles bien définies. Les ingénieurs font de leur mieux pour compenser les imperfections liées au processus de fabrication et aux propriétés des composants électroniques eux-mêmes (interférences, bruit thermique, et ainsi de suite) afin de garantir que le système suive toujours la même logique et fonctionne correctement. Lorsque les composants refusent d'agir comme on l'espère (saturation, lenteur), nous rejetons la faute sur l'ordinateur.

La cohérence des machines et leurs merveilleuses capacités de calcul font de l'ordinateur un outil si intéressant pour ceux qui parviennent à le contrôler et à le maîtriser. Bien sûr, il faut admettre que tout n'est pas parfait et les personnes qui se plaignent du manque de fiabilité des ordinateurs n'ont pas entièrement tort. En dépit du fonctionnement parfait des composants, les programmes informatiques eux-mêmes se comportent mal dans différentes situations. En effet, même si le matériel informatique est souvent cohérent et fiable, il est impossible de prévoir le comportement d'un seul programme informatique complexe sur le long terme et, *a fortiori*, d'un ensemble de logiciels interdépendants (comme c'est le cas d'un système d'exploitation). Cela rend difficile la validation d'un programme, même dans l'hypothèse où l'on parvienne à établir un modèle détaillé suffisamment strict et exempt de défauts du fonctionnement du programme. Pourquoi ? En 1936, Alan Turing, le père de la programmation actuelle, a prouvé par l'absurde qu'il ne peut pas y avoir de méthode *générale* pour déterminer le résultat d'une procédure informatique ou d'un algorithme dans un temps défini (bien qu'il puisse exister *certaines* méthodes spécifiques pour *certains* algorithmes)[1].

En pratique, si on ne peut donc pas espérer qu'un système d'exploitation ou un traitement de texte se comporte toujours exactement de la façon prévue par son auteur ou son utilisateur, on peut tout de même s'attendre à ce que le même programme installé sur deux systèmes disposant des mêmes composants matériels ait un comportement cohérent et identique si on lui fournit les mêmes données en entrée (à moins bien sûr qu'un des deux systèmes ne soit écrasé par un piano ou influencé par d'autres événements externes). Cela constitue bien sûr une excellente nouvelle pour les entreprises qui commercialisent les programmes, mais les personnes en charge de la sécurité préféreraient parfois que les ordinateurs soient un peu moins déterministes. Pas nécessairement tant dans la façon dont ils se comportent que dans les résultats qu'ils produisent.

Prenons l'exemple du cryptage des données et en particulier le concept de cryptographie à clé publique. Cette brillante et nouvelle forme de chiffrement (entre autres) fut inventée dans les années 1970 par Whitfield Diffie et Martin Hellman, puis mise en application peu de temps après par Ron Rivest, Adi Shamir et Len Adleman. Ce système, aussi appelé cryptographie asymétrique, repose sur un concept simple : certaines choses sont plus difficiles que d'autres. Cela paraît évident, bien sûr, mais ajoutez quelques notions mathématiques de haut niveau et vous obtenez une invention de premier plan.

La cryptographie traditionnelle, ou symétrique, repose sur une information "secrète" (la clé) connue de toutes les personnes impliquées. La clé est nécessaire mais suffisante pour chiffrer et ensuite déchiffrer l'information transmise, si bien qu'une personne extérieure, même si elle connaît la méthode de chiffrement utilisée, ne peut pas découvrir le contenu du message. Mais, comme il est nécessaire de partager un secret, cette approche est assez peu pratique en terme de communication informatique. En effet, il est d'abord nécessaire d'établir un moyen sécurisé pour échanger le secret avant même de communiquer – transférer le secret sur un canal non sécurisé rendrait le système vulnérable au déchiffrement. En informatique, on communique souvent avec des systèmes ou des personnes que l'on n'a jamais vus auparavant et avec lesquels on ne dispose d'aucun autre moyen sécurisé pour entrer en contact.

La cryptographie à clé publique, au contraire, ne repose pas sur un secret partagé. Chaque partie dispose de deux éléments : le premier (la clé publique) permet de créer un message chiffré mais ne sert quasiment à rien pour le décrypter ; le second (la clé privée) s'utilise pour décrypter le message chiffré. Les personnes qui communiquent peuvent échanger leurs clés publiques sur un canal non sécurisé, même s'il est surveillé. Elles se fournissent mutuellement l'information (inutile pour un observateur externe) nécessaire au chiffrement des messages qu'elles s'échangent, mais elles conservent pour elles-mêmes la partie nécessaire au déchiffrement des données. Tout à coup, la communication sécurisée entre deux parfaits inconnus (un client assis sur le canapé de son appartement et le serveur d'un site de vente en ligne) devient une réalité.

Le système de chiffrement à clé publique RSA (Rivest, Shamir et Adleman), original, repose sur le fait que multiplier deux nombres de grande taille représente une complexité de calcul assez faible, puisque directement proportionnelle au nombre de chiffres à multiplier. À l'inverse, il est beaucoup plus difficile de décomposer en produit de facteurs premiers (factorisation) un nombre de grande taille, à moins d'être un génie de la cryptographie travaillant pour la NSA, et encore. L'algorithme RSA choisit tout d'abord deux nombres premiers* de très grande taille, p et q, et les multiplie. Il utilise ensuite ce produit et un autre nombre premier**, $(p–1)(q–1)$, pour créer une clé publique.

---

* Un nombre premier est un entier positif qui ne se divise que par 1 ou lui-même.

** Le second entier n'a d'autre facteur commun avec le premier entier que 1 et –1 (leur plus grand commun diviseur est 1).

Cette clé est utilisée pour chiffrer l'information mais ne suffit pas pour la déchiffrer sans recourir à la factorisation.

La factorisation des produits de deux nombres premiers de grande taille est souvent impossible, ce qui protège des attaques. L'algorithme de factorisation le plus rapide sur les ordinateurs traditionnels, le crible général de corps de nombres (GNFS), aurait besoin de plus de mille ans pour factoriser un nombre entier de 1 024 bits, à raison d'un million de tests par seconde. En revanche, il ne faut que quelques secondes à un PC de moyenne gamme pour trouver deux chiffres premiers dont le produit soit de cette taille.

En outre, dans le système RSA, on crée également une clé privée en plus de la clé publique. Cette clé privée contient sur les nombres premiers des données supplémentaires qui peuvent être utilisées pour décrypter toute information chiffrée avec la clé publique. L'astuce est possible, grâce au théorème des restes chinois, le théorème d'Euler et d'autres concepts mathématiques fascinants que certains lecteurs curieux auront peut-être envie de découvrir par eux-mêmes[2].

D'autres systèmes de cryptographie à clé publique fondés sur des formules mathématiques complexes ont également été conçus par la suite (comme la cryptographie utilisant les courbes elliptiques, par exemple), mais tous partagent le principe de clés publiques et de clés privées. Cette méthode s'est révélée sûre pour sécuriser les e-mails, les transactions sur le Web, même lorsque l'échange s'effectue entre deux parties qui ne sont jamais entrées en contact et ne disposent pas d'un canal sécurisé pour échanger des informations complémentaires avant d'établir la connexion*. Quasiment tous les protocoles de communication sécurisée utilisés aujourd'hui, comme SSH (Secure Shell) ou SSL (Secure Sockets Layer) pour marquer numériquement des mises à jour ou des cartes à puce, existent grâce aux découvertes de Diffie, Hellman, Rivest, Shamir et Adleman.

Générer automatiquement des nombres aléatoires

Il subsiste un problème : lors de l'implémentation du système RSA sur une machine déterministe, la première étape consiste à générer deux nombres premiers de très grande taille, p et q. Un ordinateur trouve facilement un premier de grande taille, mais ces nombres doivent également être impossibles à deviner par d'autres machines et ne doivent être les mêmes sur tous les ordinateurs (s'ils l'étaient, une attaque de l'algorithme ne nécessiterait aucune factorisation puisque p et q seraient connus par toute personne disposant d'un ordinateur équivalent).

---

* Pour être complet, il faut noter que la cryptographie à clé publique est entre autre vulnérable aux attaques menées par une personne qui créé et fournit une clé publique fausse de façon à intercepter les communications. Pour se prémunir de ce type d'attaques, des méthodes supplémentaires doivent être mis en place pour vérifier l'authenticité de la clé, soit en concevant un échange sécurisé, soit en établissant une autorité centrale qui vérifie et valide les clés (infrastructures à clés publiques, PKI ou IGC).

De nombreux algorithmes ont été développés ces dernières années pour trouver rapidement des nombres premiers pouvant être utilisés (nombres pseudo-aléatoires) et effectuer aussi rapidement des tests préliminaires sur ces nombres[3]. Mais, pour générer un nombre premier vraiment impossible à prévoir, il est nécessaire d'accumuler de l'entropie ou du hasard, qu'il s'agisse de sélectionner un nombre premier parmi un ensemble donné ou de partir d'une valeur aléatoire et de choisir ensuite le premier nombre premier sur lequel on tombe.

Bien que le hasard soit essentiel au moment de la génération de la clé, cela ne suffit pas. La cryptographie à clé publique repose sur des calculs complexes et souffre donc d'une grande lenteur, en particulier lorsqu'on la compare à la cryptographie symétrique, qui, elle, utilise des clés courtes que les machines déchiffrent à l'aide de calculs connus pour s'exécuter rapidement.

Pour implémenter un protocole comme SSH, dont on attend un certain niveau de performances, il est préférable d'établir une communication initiale, d'effectuer une vérification basique à l'aide d'algorithmes à clés publiques et, donc, de créer un canal sécurisé. La seconde étape consiste à échanger une clé de chiffrement symétrique et compacte, par exemple de 128 bits, puis de continuer la communication en utilisant le système de cryptographie symétrique. Le principal défaut de la cryptographie symétrique est donc résolu par la création initiale (et donc lente) d'un canal sécurisé pour échanger le secret à partager. Ensuite, en passant à des algorithmes plus rapides, l'utilisateur peut alors bénéficier de meilleures performances sans sacrifier la sécurité. Cependant, pour utiliser la cryptographie symétrique correctement, il est toujours nécessaire de recourir à un certain degré d'entropie afin de générer une clé de session symétrique imprévisible pour chaque communication sécurisée..

La sécurité des générateurs de nombres pseudo-aléatoires

Les programmeurs ont conçu de nombreuses méthodes pour que les ordinateurs génèrent des nombres apparemment aléatoires. Ces algorithmes sont regroupés sous le nom générique de générateurs de nombres pseudo-aléatoires (PRNG).

Les générateurs de nombres pseudo-aléatoires suffisent pour des applications simples, comme la création d'événements "aléatoires" dans des jeux vidéo ou les titres des messages non sollicités dans le cadre d'un mailing de masse. Prenons le générateur de congruence linéaire (GCL)[4], un exemple classique de ce type d'algorithme. En dépit de son nom obscur, ce générateur de nombres aléatoires effectue une suite de calculs simples (multiplication, addition et modulo*) chaque fois qu'il génère une sortie

---

* L'opérateur modulo renvoie le reste entier de la division de deux nombres. Par exemple, 7 divisé par 3 donne un résultat de 2 et un reste de 1 (7=2*3+1). 7 modulo 3 est donc égal à 1.

"aléatoire". Ce générateur utilise sa sortie précédente r_t pour calculer la valeur de la sortie suivante r_{t+1} (où t indique le temps).

$$r_{t+1} = (a \times r_t + c) \bmod M$$

L'opérateur modulo empêche que le résultat dépasse la plage prédéfinie de valeurs possibles. Si r_0, a, M et c, un ensemble de variables de contrôle pour le générateur, sont tous des entiers positifs, tous les résultats de l'équation sont compris entre 0 et $M–1$.

Pourtant, alors que la sortie de cet algorithme peut, avec quelques réglages, afficher des propriétés statistiques qui le rendent utilisable pour générer des nombres pseudo-aléatoires, rien n'est vraiment imprévisible dans les calculs effectués. Et là réside le problème : un attaquant peut facilement développer sa propre copie du générateur et l'utiliser pour connaître tous les résultats que notre générateur créera. Même si le générateur part d'un état initial (r_0) inconnu de l'attaquant, il lui est souvent possible de déduire des propriétés importantes de cette valeur en observant les sorties produites par le générateur de la victime puis d'utiliser ce qu'il sait pour régler sa version du générateur afin qu'elle imite la nôtre. En fait, une méthode générale pour reconstruire et prédire tous les polynômes des générateurs de congruence a été établie dans les années 1990[5]. Il est sage de ne pas ignorer ce petit inconvénient car il crée un trou béant dans cet algorithme lorsque son utilisation est essentielle à la sécurité.

On a compris peu à peu que la seule méthode qu'un ordinateur peut employer pour produire des données quasiment imprévisibles (à moins que la mémoire ne subisse une panne grave ou que le processeur ne fonde) consiste à rassembler autant d'informations non prévisibles que possible à partir de son environnement physique et de les utiliser comme une valeur à transférer à tous les programmes qui nécessitent une bonne part de hasard. Le problème tient au fait qu'un ordinateur n'a pas de "sens" qui lui permette de sonder son environnement à la recherche de signaux externes apparemment aléatoires. Néanmoins, il existe une méthode assez intéressante pour contourner ce problème.

Entropie entrée-sortie : votre souris vous parle

Sur presque tous les systèmes informatiques, les périphériques externes communiquent des événements asynchrones, ces informations provenant de la carte réseau ou du clavier à l'aide d'un mécanisme d'interruption matériel. Chaque périphérique dispose d'un nombre d'interruptions matérielles (IRQ, de l'anglais *Interrupt Request*) et indique les événements importants en changeant le voltage sur la ligne d'entrée-sortie matérielle de l'élément qui correspond à cet IRQ en particulier dans l'ordinateur. La modification est alors interprétée par un périphérique appelé *le contrôleur programmable d'interruption* (PIC, de l'anglais *Programmable Interrupt Controller*) qui délivre les appels d'interruption au processeur (ou aux processeurs).

Le processeur indique au contrôleur d'interruption si, quand, comment et dans quel ordre de priorité il doit signaler les appels des périphériques à l'unité centrale, ce qui permet au processeur de gérer les événements efficacement et de façon fiable. Lorsqu'il reçoit un signal du contrôleur d'interruption, le processeur interrompt provisoirement la tâche qu'il effectue, à moins qu'il ait choisi d'ignorer toute interruption à ce moment (au cas où il est particulièrement occupé). Il invoque ensuite un code assigné par le système d'exploitation pour gérer l'appel du périphérique ou du groupe de périphériques. Une fois que le programme gère l'événement, le processeur restaure le processus original et son contexte – l'état de l'environnement au moment de l'interruption – et continue son exécution comme si rien ne s'était produit.

Interruptions : un exemple pratique

En pratique, de nombreuses étapes supplémentaires se déroulent lors de la détection d'une condition externe, la génération et la réception d'un IRQ. La Figure 1.1 illustre les événements qui sont déclenchés par l'appui ou le relâchement d'une touche du clavier. Avant même que vous n'appuyiez sur une touche, une puce située dans le clavier sert de microcontrôleur et surveille en permanence tout changement de l'état du clavier.

Le clavier est organisé comme un tableau de câbles horizontaux et verticaux. Les touches (microcontacts ou membranes sensibles à la pression) sont installées à l'intersection de chaque rangée et colonne. Le contrôleur teste chaque rangée et chaque colonne séparément à une vitesse très élevée.

Si le contrôleur du clavier détecte par exemple un circuit fermé lorsqu'il teste l'intersection de la rangée 3 et de la colonne 5 (ce qui est indiqué par une résistance faible lorsque le voltage est appliqué à ces lignes), il en conclut que la touche à cet emplacement (J) est enfoncée. Lorsque le contrôleur du clavier sent une modification, il convertit les coordonnées de la rangée et de la colonne en scan code, une valeur qui identifie chaque touche. L'information est ensuite placée dans la mémoire tampon interne d'une puce qui signale au processeur principal l'existence de nouvelles données, puis reprend sa tâche.

La puce du contrôleur d'entrée du clavier est l'équivalent du contrôleur de la carte mère. Le contrôleur d'entrée gère habituellement tous les périphériques d'entrée élémentaires, comme la souris et le clavier. Il reçoit un scan code unique de la puce du clavier et signale l'interruption adéquate au contrôleur d'interruption. Dès que ce dernier détermine qu'il peut envoyer cette IRQ en particulier, il transfère le signal au processeur, qui généralement interrompt sa tâche courante et fait appel au gestionnaire d'interruption du système d'exploitation. Ce gestionnaire est censé lire les données et indiquer à la puce qu'il a effectué la lecture du scan code avec succès. Le contrôleur d'entrée reprend alors

ses activités normales et lit éventuellement un autre scan code en provenance du clavier si de nouvelles données se trouvent dans la mémoire tampon*.

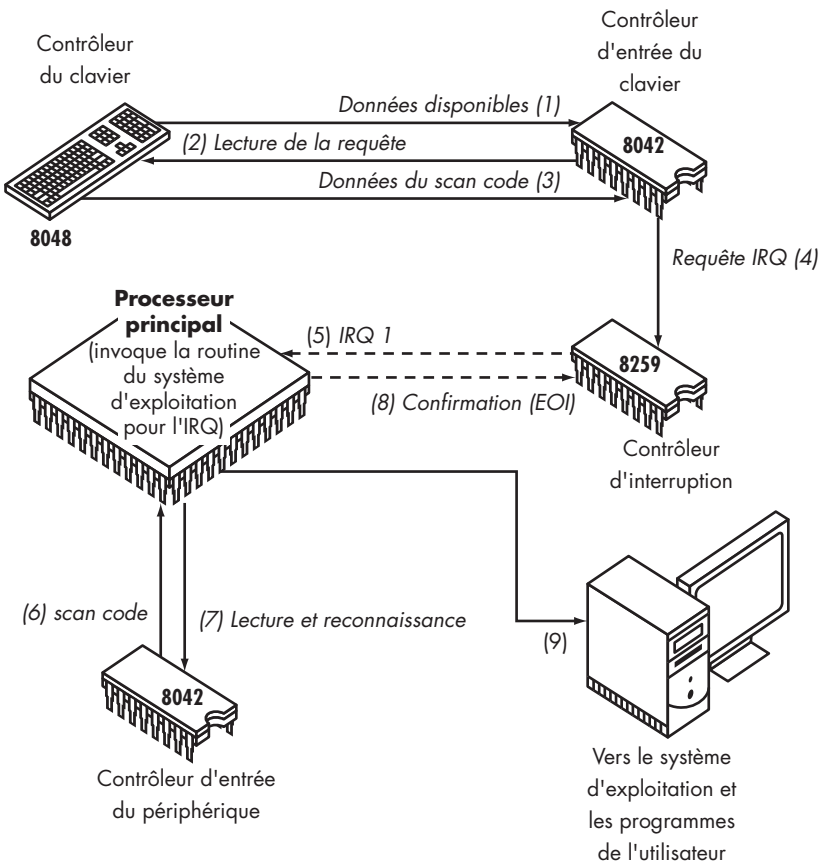


Figure 1.1

Communications du clavier à l'ordinateur.

Bien qu'indirectement, ce schéma de fonctionnement est important pour la création de nombres aléatoires. L'ordinateur, en utilisant le système de notification asynchrone des événements (interruptions), reçoit presque instantanément des indications précises sur

* Sur de nombreuses architectures, il est nécessaire d'indiquer au contrôleur d'entrée que l'interruption a été traitée et qu'il ne doit plus bloquer les interruptions suivantes. C'est le rôle du code de fin d'interruption EOI (de l'anglais End of Interrupt).

l'activité de l'utilisateur et, ce qui est peut-être plus intéressant, mesure précisément le délai entre chaque pression sur les touches. Bien que cette information ne soit pas totalement imprévisible, elle constitue sans doute le meilleur signal externe, quantifiable et quelque part non déterministe, que la machine peut obtenir. Et, donc, pour contourner la nature déterministe de l'ordinateur et insérer une part de hasard dans ses calculs, les auteurs de l'implémentation de générateurs de nombres pseudo-aléatoires créent de l'entropie à partir du comportement généralement imprévisible de certains périphériques comme la souris, le clavier, les interfaces réseau et parfois les disques durs. Pour cela, ils ajoutent dans le gestionnaire d'interruption du système d'exploitation un code qui enregistre certains paramètres pour chaque événement utilisable.

On peut bien sûr considérer qu'aucune de ces sources ne fournit tout le temps de données vraiment aléatoires. Si l'utilisateur tape "camésc", il est plus que vraisemblable que les lettres qui suivent seront "ope". Mais certains comportements, comme le choix du mot caméscope de notre exemple, sont en fait assez imprévisibles d'un point de vue pratique (sans entrer dans une discussion philosophique sur le libre arbitre et les univers déterministes). En fait, cette méthode pour ajouter de l'entropie fonctionne raisonnablement bien car elle intègre plusieurs facteurs qui ne peuvent pas être pris en compte, surveillés ou prédits par un attaquant sans qu'il perde sa santé mentale. Selon les lois de la probabilité, si on collecte des données de toutes ces sources pendant une durée assez longue, on obtient obligatoirement une certaine part de hasard. En collectant les informations d'une mémoire tampon, on crée un stock d'entropie qui peut être plus ou moins important en fonction du nombre de données imprévisibles fournies ou utilisées. Malheureusement, ces petites portions de hasard dans le stock (lorsque les données que nous entrons au clavier sont le fruit d'événements cosmiques) s'accompagnent toujours de données hautement prévisibles et ne peuvent donc pas être utilisées en l'état pour générer des nombres aléatoires.

Pour s'assurer que le taux d'entropie des données collectées lors du processus de gestion et de remplissage de notre stock soit également réparti dans tous les bits en sortie du générateur de nombres pseudo-aléatoires, ce stock doit être haché. Autrement dit, il doit être soigneusement mélangé pour qu'aucune partie des données ne soit plus facile à prédire qu'une autre. Chaque bit en sortie doit dépendre à part égale de tous les bits en entrée. Il peut être difficile d'atteindre ce but sans savoir quelles parties de l'information sont prévisibles ou non (l'information qui ne peut pas facilement être obtenue en surveillant les touches du clavier ou les mouvements de la souris).

Fonctions de résumé

Heureusement, les fonctions de hachage sécurisées, un élément phare de la cryptographie moderne, nous permettent de mélanger les données afin d'obtenir plus d'entropie dans chaque bit en sortie, quel que soit le manque d'uniformité des données en entrée.

Ces fonctions génèrent un résumé de longueur fixe, autrement dit un identificateur unique pour un bloc de données de taille arbitraire en entrée. Mais ce n'est pas tout.

Toutes les fonctions de hachage possèdent deux propriétés importantes :

- Il est aisé de calculer le résumé mais il n'est pas possible de déduire le message original ou l'une de ses propriétés à partir de ce résultat. Tout changement apporté au message peut autant affecter toutes les propriétés de la sortie que n'importe quelle autre modification.
- La possibilité que deux messages distincts aient le même résumé dépend uniquement de la taille du résumé. Si sa taille est suffisamment importante pour que des recherches exhaustives soient impossibles (entre 128 et 160 bits, soit un nombre de combinaisons compris entre 3.4E+38 à 1.46E+48), il n'est pas possible que deux messages aient le même résumé.

Les fonctions de hachage fournissent donc un moyen de répartir uniformément dans les données en sortie la part de hasard présente dans les données en entrée. Cela résout le problème du hasard créé à partir de sources locales dont l'entropie est prévisible. En effet, nous collectons un nombre approximatif de données aléatoires depuis l'environnement que nous mélangeons avec des données prévisibles, puis nous générons un résumé qui est assuré d'être aussi imprévisible que l'entropie récoltée en premier lieu, quelle que soit la manière dont cette part de hasard est répartie dans les données en entrée.

Comment fonctionnent les fonctions de hachage ? Là encore, certaines reposent sur des problèmes mathématiques qui sont, autant qu'on le sache, très difficiles à résoudre. En fait, tout algorithme de cryptographie symétrique ou à clé publique peut être converti en fonction sécurisée de hachage. Aussi longtemps que personne ne découvre une meilleure solution, cette méthode convient parfaitement.

Cependant, cette méthode implique l'utilisation d'outils lents et très complexes pour générer des résumés, ce qui la rend souvent peu pratique à implémenter, en particulier lorsqu'il s'agit de l'intégrer dans un système d'exploitation. Une autre solution consiste à traiter les données de façon que l'interdépendance entre tous les bits en entrée et en sortie soit suffisamment complexe pour que l'obfuscation du message en entrée soit totale, en espérant que "cela suffise" à empêcher les techniques connues de cryptanalyse. Cette approche empirique est raisonnable puisqu'une bonne partie de l'informatique repose en fait sur la devise "espérons que cela suffise".

Les groupes d'algorithmes alors utilisés, comme les fonctions MD2, MD4, MD5 et SHA-1, présentent l'avantage d'être plus rapides et simples à utiliser que leurs équivalents fondés sur des formules mathématiques complexes mais également, lorsqu'ils sont bien conçus, de résister aux techniques traditionnelles de cryptanalyse. Leur faiblesse tient au fait qu'on ne peut pas prouver qu'ils soient sécurisés, puisque aucun d'entre eux ne peut se réduire à une formule classique et dure à résoudre. D'ailleurs, certains chercheurs ont prouvé que ces algorithmes présentaient des failles[6].

Comme nous l'avons déjà indiqué, les fonctions de hachage rendent un grand service aux générateurs de nombres pseudo-aléatoires dans le sens où elles peuvent être exécutées sur un segment de données qui contient *n* bits aléatoires et n'importe quelle quantité de bits prévisibles afin de créer une empreinte qui contiendra *n* bits aléatoires répartis également sur tous les bits de ce résumé (grâce aux deux propriétés fondamentales des fonctions de hachage que nous avons décrites). La fonction de hachage est donc un extracteur pratique d'entropie. Si la fonction de hachage utilise un nombre suffisant de données imprévisibles récoltées depuis un gestionnaire d'interruption, il est possible de générer des nombres aléatoires sans divulguer de renseignements sur la forme exacte de l'information utilisée pour les créer, tout en évitant le risque que des données imparfaites en entrée puissent affecter la sortie de manière significative. Pour ne pas compromettre l'ensemble du processus, il suffit de s'assurer que la quantité d'entropie soit suffisamment importante et de fournir à la fonction de hachage cet amas de données d'interruption. Si l'attaquant peut prévoir une grande partie des données utilisées pour générer le nombre aléatoire et que le reste ne représente qu'une poignée de combinaisons possibles, il peut alors lancer une attaque par force brute sur l'implémentation en essayant simplement toutes les valeurs possibles. Si nous utilisons par exemple une fonction de hachage qui crée une empreinte de 128 bits, quelle que soit la quantité de données collectées (200 octets ou 2 mégaoctets de pression sur le clavier), nous devons être sûrs qu'au moins 128 de ces bits en entrée sont imprévisibles pour un attaquant avant d'utiliser la fonction de hachage.

De l'importance d'être pédant

Prenons l'exemple d'un utilisateur qui décide d'écrire un script shell alors que le stock d'entropie est vide, peut-être parce qu'une précédente opération a nécessité l'utilisation de nombres aléatoires auparavant. L'attaquant note que l'utilisateur écrit un script car `vi delallusers.sh` est exécuté. Il considère ensuite que le début du script doit commencer aux lignes `#!/bin/sh`. Bien qu'il ne puisse pas être sûr de ce qui suit, il peut s'attendre à ce que le script s'ouvre lors de l'utilisation d'une commande shell et qu'il ne s'agisse pas d'une ode aux tamanoirs.

À ce moment, un utilitaire de chiffrement demande soudainement au système un nombre aléatoire de 128 bits pour l'utiliser comme clé de session et ainsi protéger les communications. En revanche, le système ne parvient pas à estimer correctement la quantité d'entropie disponible dans la mémoire tampon qui a enregistré le processus d'écriture des premières lignes du script, si bien que la tâche de l'attaquant est maintenant facile. L'ordinateur ne dispose pas de l'information nécessaire pour savoir si l'action effectuée par l'utilisateur à ce moment précis est prévisible par d'autres ou non. Il ne peut que spéculer (aidé par les hypothèses faites par le programmeur) sur le fait que les actions de l'utilisateur au fil des minutes ou des heures finiront par se résumer à quelque chose qui

ne peut pas être prévu avec précision et qu'en moyenne cette quantité de données entrées dépendra de facteurs que l'attaquant ne peut pas prévoir.

L'attaquant, à ce point, connaît la plus grande part du contenu du stock d'entropie et ne doit plus choisir que parmi quelques milliers d'options pour la partie qu'il ne connaît pas (en dépit du fait que le système d'exploitation est persuadé que la quantité d'entropie dans la mémoire tampon est beaucoup plus importante). Ces milliers d'options ne représentent pas un gros défi pour une personne assistée d'un ordinateur. Par conséquent, au lieu d'avoir un nombre aléatoire de 128 bits et donc une combinaison de 39 chiffres, une application de cryptographie peu sûre génère un nombre à partir de données en entrée qui se résument à quelques milliers d'options. L'attaquant peut alors facilement tester ces options les unes après les autres et donc déchiffrer rapidement les informations qui étaient censées rester sécurisées.

L'entropie ne doit pas être gâchée

Comme il est presque impossible de prévoir précisément la quantité d'entropie collectée par l'utilisateur dans une période courte et afin d'éviter que la sortie du PRNG soit prévisible, toutes les implémentations intègrent le résumé ou l'état interne du générateur de nombres pseudo-aléatoires lors de la création de la nouvelle sortie. La sortie précédente devient une part de l'équation utilisée pour calculer la valeur suivante du PRNG.

Dans ce cas, une fois une quantité suffisante d'entropie rassemblée dans le système, les données les plus récentes utilisées pour remplir de nouveau le stock d'entropie n'ont pas besoin d'être totalement aléatoires à tout moment pour garantir une sécurité élémentaire.

Il y a cependant un autre problème. Si les implémentations s'exécutent pendant une durée prolongée en se fondant sur l'ancienne entropie héritée, en répétant simplement plusieurs fois des fonctions de hachage MD5 ou SHA-1, la sécurité ne repose plus alors que sur l'algorithme de hachage. Or nous avons vu que celui-ci ne peut pas être totalement fiable en termes de performance et de sécurité des communications. En outre, les fonctions de hachage n'ont pas forcément été évaluées par des cryptographes compétents pour savoir si elles sont adaptées à cet usage précis. L'implémentation ne repose plus uniquement sur les propriétés de hachage des bits de la fonction mais dépend maintenant complètement de son invulnérabilité aux attaques. Si une petite portion d'information sur l'état interne du générateur est dévoilée à chaque étape et qu'aucune nouvelle donnée imprévisible n'est ajoutée au stock, ces données peuvent à long terme permettre de reconstruire ou de deviner avec une quasi-certitude l'état interne. Il est alors possible de prévoir le comportement futur du périphérique. En revanche, si de nouvelles données aléatoires sont ajoutées à un rythme qui, du moins statistiquement, empêche une réutilisation significative de l'état interne, l'attaque devient beaucoup moins réalisable même si la fonction de hachage est cassée.

De nombreux experts déconseillent d'accorder un tel niveau de confiance et de dépendance envers la fonction de hachage pour les applications les plus exigeantes. Il est donc important que l'implémentation conserve la trace de la quantité estimée d'entropie que le système a collectée, ce qui, même si cette estimation n'est pas exacte momentanément, reflète une tendance statistique générale conforme à ce que nous attendons des sources utilisées. À court terme, des fluctuations mineures dans la disponibilité de l'entropie externe peuvent se produire, comme dans le script d'exemple dont nous avons parlé, et sont alors compensées par l'algorithme qui réutilise la sortie précédente. Néanmoins, il est nécessaire d'effectuer des prévisions à long terme précises afin de garantir la reconstitution fréquente du stock d'entropie et de minimiser l'exposition au cas où la fonction de hachage dévoile l'état interne au fil du temps. En tant que telle, l'implémentation doit tenir compte de l'entropie dépensée dans les données fournies aux processus utilisateur et refuser de fournir plus de nombres aléatoires jusqu'à ce qu'une quantité suffisante d'entropie soit disponible.

Le système conçu et mis en œuvre par Theodore Ts'o en 1994 au MIT (Massachusetts Institute of Technology) représente un bon exemple d'implémentation de PRNG qui tienne compte de toutes ces notions. Son mécanisme, /dev/random, a tout d'abord été implémenté dans Linux puis introduit dans des systèmes comme FreeBSD, NetBSD et HP/UX. Il contrôle un certain nombre d'événements en entrée et en sortie (I/O), mesure les intervalles de temps entre eux ainsi que d'autres caractéristiques importantes des interruptions. Il préserve également le stock d'entropie lors de la fermeture du système en le sauvegardant sur disque, ce qui empêche le système de démarrer dans un état totalement prévisible et le rend plus difficile à attaquer.

Attaques : les implications d'un changement soudain de paradigme

Quel problème pourrait poser ce système apparemment infaillible pour la fourniture de nombres aléatoires imprévisibles aux applications exigeantes ? Aucun, du moins pas là où vous vous attendriez à en trouver. Les chiffres générés sont en effet difficiles à prédire.

Le raisonnement de l'auteur de cette technologie contient cependant une erreur légère mais désastreuse. La conception de M. Ts'o suppose en effet que l'assaillant cherche à prévoir les nombres aléatoires en fonction de ce qu'il sait de la machine et de son environnement. Mais que se passe-t-il si l'attaquant veut faire tout le contraire ?

L'attaquant qui dispose d'un compte sur la machine, même sans avoir d'accès direct à l'information que l'utilisateur tape, peut déduire le moment exact auquel l'activité en entrée survient dans le système en vidant le stock d'entropie (ce qu'il peut réaliser simplement en demandant des données aléatoires au système sans les conserver) puis en surveillant la disponibilité de la sortie du PRNG. S'il n'y a pas d'activité en entrée/sortie, le PRNG

ne dispose pas de nouvelles données disponibles, puisque l'estimation de l'entropie ne change pas. Si une ou plusieurs touches sont enfoncées, une petite quantité d'informations sera disponible pour l'attaquant, qui peut alors en déduire que l'une des touches a été enfoncée ou relâchée.

D'autres événements, comme l'activité du disque, génèrent également une sortie du PRNG, mais les motifs dans le temps et la quantité de l'entropie collectée de cette façon diffèrent des caractéristiques des données d'interruption transmises par un clavier. Ainsi, il est possible et facile de distinguer les événements en fonction de la quantité de données disponibles à un moment donné. Les données provenant des touches du clavier seront différentes des données relatives à l'activité du disque.

En fin de compte, une méthode conçue pour sécuriser au maximum la génération de nombres aléatoires entraîne en fait une dégradation de la vie privée de l'utilisateur : il est possible de tromper ce mécanisme qui estime la quantité d'entropie disponible auprès d'une source externe et de l'utiliser pour contrôler certains aspects des activités en entrée sur le système. Même si l'attaquant ne peut pas détecter exactement ce qui est tapé, la frappe de certains mots présente certains motifs dans le temps identifiables, en particulier s'il dispose de l'information concernant l'appui ou le relâchement d'une touche en particulier, comme c'est le cas ici. En examinant ces motifs, l'attaquant peut déduire de quelle entrée il s'agit ou tout au moins la deviner plus facilement.

Examen plus détaillé des motifs dans le temps en entrée

Une analyse approfondie menée par une équipe de chercheurs de l'université de Californie[7] indique qu'il est possible de déduire certaines propriétés des entrées de l'utilisateur, voire de complètement reconstruire les données, en se concentrant uniquement sur la fréquence de frappe entre deux touches. Cette étude a conclu qu'il pouvait exister des variations entre une personne qui tape lentement et un opérateur clavier compétent mais que la vitesse de frappe entre deux touches suivait généralement des motifs dans le temps.

En effet, non seulement nous disposons nos mains d'une certaine façon, mais la disposition des touches sur le clavier affecte également la vitesse à laquelle nos doigts peuvent atteindre une touche. Par exemple, le laps de temps entre les touches e et n est généralement différent de l'intervalle entre b et j. Dans le premier cas, comme une main contrôle le côté gauche du clavier et l'autre, le côté droit (voir Figure 1.2), la frappe des deux lettres ne demande quasiment aucun mouvement, si bien que les deux touches sont enfoncées presque simultanément, avec un intervalle de moins de 100 millisecondes. En revanche, la frappe consécutive de b et j est plus complexe car cela exige d'utiliser deux fois la même main, ce qui prend beaucoup plus de temps.

Figure 1.2

La zone du clavier généralement dévolue à chaque main. Les touches grises sont habituellement contrôlées par la main gauche et les touches blanches, par la main droite.

Après avoir mené un certain nombre d'expériences, les auteurs de cette étude estiment qu'environ 1,2 bit d'information par touche enfoncée peut être obtenue à partir de la fréquence de la frappe. En observant le rythme des séquences, il est possible de déterminer l'ensemble des entrées du clavier les plus susceptibles de générer ce schéma et donc de deviner plus facilement la chronologie exacte des touches enfoncées. L'idée de compter des fractions de bit peut sembler ridicule, mais cela signifie que le nombre de possibilités pour chaque touche peut être réduit par $2^{1,2}$, soit environ 2,40 fois. Pour une simple touche, qui ne véhicule généralement pas plus de 6 bits de hasard, cela réduit le nombre d'options possibles de 64 éléments à environ 26.

Comme vous pouvez le voir, il existe bien une manière de limiter le nombre de possibilités si nous voulons deviner quelles sont les touches enfoncées. Même si cela peut ne pas être particulièrement impressionnant en soi, il faut également considérer le fait que les données saisies au clavier ont peu de chances d'être une suite de lettres sans queue ni tête. On estime que l'entropie d'un texte en anglais est de 0,6 à 1,3 bit par caractère seulement[8], ce qui signifie qu'environ 1,5 à 2,5 tentatives en moyenne suffisent pour réussir à prévoir le caractère suivant. Avec une méthode qui réduise encore davantage le champ de recherche, il est possible de découvrir sans ambiguïtés les mots qui correspondent à presque toutes les données entrées.

Pour vérifier leur estimation et la démontrer dans la pratique, les chercheurs ont utilisé le modèle de Markov caché et l'algorithme de Viterbi pour deviner les touches enfoncées. On appelle modèle de Markov une méthode qui décrit un système d'état discret dont la prochaine valeur dépend uniquement de son état actuel, et non pas des valeurs précédentes (chaîne de Markov). Le modèle de Markov caché est une variante qui fournit une méthode pour décrire un système dans lequel chaque état interne génère une observation, mais pour laquelle l'état actuel n'est pas connu. Ce modèle est couramment utilisé dans des applications comme la reconnaissance vocale, lorsqu'on cherche à obtenir des

données pures (une représentation textuelle de la parole) à partir de ses manifestations spécifiques (l'onde sonore échantillonnée).

Les auteurs de l'étude ont conclu que le modèle de Markov caché peut être utilisé pour analyser la frappe, et ils considèrent que l'état interne du système représente l'information sur les touches enfoncées, tandis que le laps de temps entre la frappe de deux touches correspond à l'observation figurant dans le modèle de Markov.

Il est possible de considérer qu'il s'agit là d'une simplification exagérée, puisque la dépendance peut être plus importante, en particulier dans la situation qu'illustre la Figure 1.3.

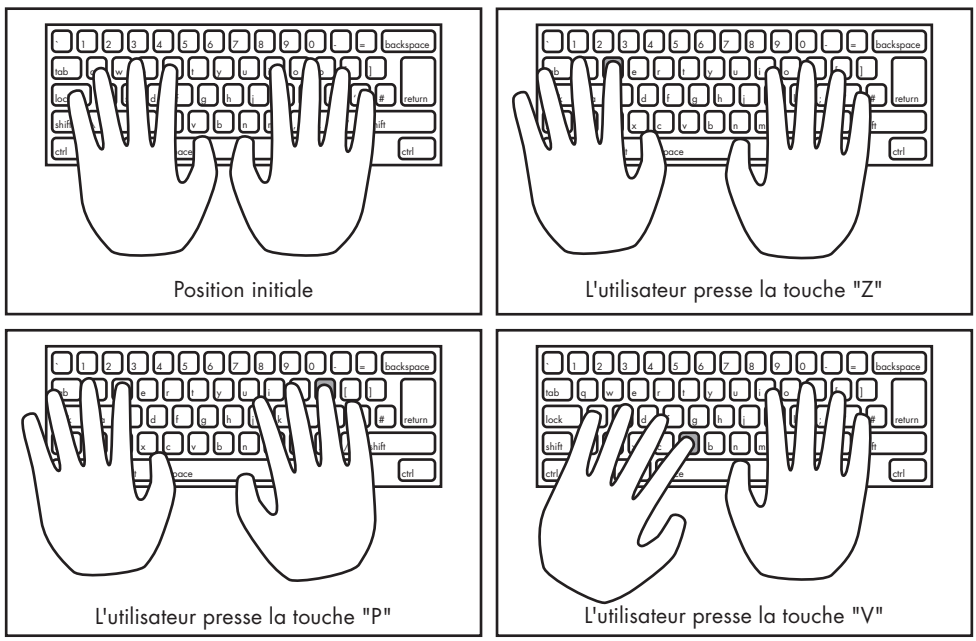


Figure 1.3

La nécessité de déplacer la main gauche dans une position différente lors de la première étape affecte ensuite le laps de temps entre les touches P et V. Or le modèle de Markov est incapable de tenir compte d'un scénario où la position de la main varie à l'étape précédente.

L'algorithme de Viterbi permet de résoudre les problèmes du modèle de Markov caché. Cet algorithme peut être utilisé pour trouver la séquence la plus probable des états internes en se fondant sur une suite d'observations. Dans ce cas particulier, on l'utilise pour déterminer la suite la plus probable de caractères à partir de l'intervalle de temps entre les touches.

L'utilisation de l'algorithme de Viterbi entraîne une réduction du champ de recherche pour les mots de passe de 8 caractères absents du dictionnaire par un facteur de 50. Dans le cas où le texte à reconstruire à partir de sa frappe est constitué de mots anglais existants dans le dictionnaire, ce facteur est susceptible d'être considérablement plus élevé.

Penchons-nous maintenant sur la surveillance des interruptions. Les recherches que nous venons de décrire se concentrent sur les informations partielles disponibles en espionnant les motifs de transfert de SSH (Secure Shell). Mais, s'il contrôle les interruptions, l'attaquant dispose alors de beaucoup plus d'informations, à savoir la durée de l'appui sur la touche, ainsi que la cadence de la frappe, puisque la durée pendant laquelle une touche est enfoncée dépend du doigt utilisé. Par exemple, l'index est le doigt qui reste généralement le moins longtemps en contact sur une touche, l'annulaire est probablement le plus lent, et ainsi de suite. Ces informations précieuses facilitent la localisation approximative des touches sur le clavier.

Par ailleurs, ces données permettent également à l'attaquant de surveiller le passage d'une main à l'autre, le moment où le premier caractère est tapé de la main gauche et le second, par la main droite, ou *vice versa*. Comme chaque main est contrôlée par un hémisphère différent du cerveau, presque tous les utilisateurs qui maîtrisent l'usage du clavier appuient souvent sur la deuxième touche avant de relâcher la première lorsqu'ils changent de main. Bien que l'enfoncement et le relâchement des touches soient impossibles à distinguer en tant que tels, un intervalle de temps particulièrement court entre deux événements clavier indique clairement ce phénomène. Dans quelques cas rares, en particulier lorsque la personne qui frappe est pressée, la seconde touche survient non seulement avant la libération mais avant même que la première touche ne soit enfoncée, ce qui se traduit par des erreurs typographiques classiques comme la frappe de "dse" au lieu de "des".

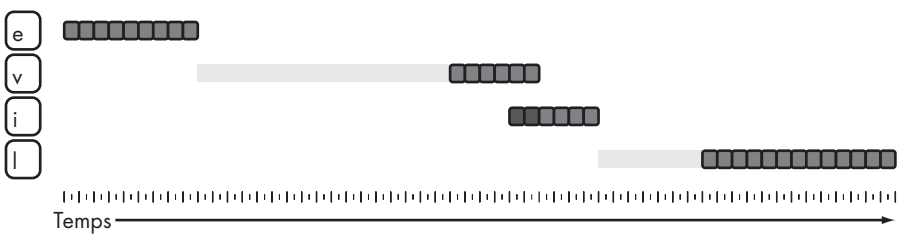


Figure 1.4

Cadence de la pression et du relâchement sur les touches lors de changement de main.

La Figure 1.4 montre un exemple d'échantillon de frappe au clavier dans le temps. L'utilisateur tape le mot *evil* ("mal" en anglais). Le majeur de la main gauche appuie sur

la touche e pendant une période de temps moyenne. Ensuite, un laps de temps important se déroule avant la frappe de la touche v car l'utilisateur doit bouger la totalité de sa main afin d'atteindre cette touche avec l'index. (Le pouce ne peut pas être utilisé car la barre Espace bloque le passage.) La touche v est enfoncée pendant une courte période de temps, comme c'est le cas de i, car ces deux touches sont enfoncées par l'index. On constate également un chevauchement : la touche i est enfoncée avant que la touche v ne soit relâchée en raison d'un changement de main. Enfin, l'annulaire enfonce la touche l après un certain temps (il n'est pas nécessaire de déplacer la main), et le contact est assez long.

Par conséquent, on peut raisonnablement penser qu'il est possible d'obtenir un taux de réussite beaucoup plus élevé dans cette attaque (la plupart de ces informations n'étaient pas disponibles dans l'exemple de l'étude originale).

Tactiques de défense immédiates

À présent que nous connaissons le potentiel de l'espionnage du clavier, comment s'en défendre ? Le meilleur moyen consiste à utiliser l'entropie d'une mémoire tampon distincte (celle du clavier) et qui ait une taille raisonnable. La mémoire tampon est vidée et transmise à la base de l'implémentation du PRNG uniquement lorsqu'elle déborde ou après un intervalle de temps beaucoup plus important que celui habituellement constaté entre la frappe de deux touches (c'est-à-dire quelques secondes au moins). On élimine ainsi la capacité de l'attaquant à mesurer la cadence de la frappe.

Avec cette solution, seuls deux types d'informations sont disponibles pour l'attaquant. La procédure de vidage de la mémoire tampon lorsqu'elle déborde révèle à l'attaquant qu'un certain nombre de touches (en fonction de la taille de la mémoire tampon) ont été enfoncées dans un laps de temps qu'il peut mesurer, mais sans que ne lui soit divulgué l'intervalle de temps exact entre chaque touche. L'attaquant peut également chronométrer une séquence du vidage de la mémoire, ce qui lui indique qu'une ou plusieurs touches ont été enfoncées pendant une durée précise mais ne lui fournit aucune information sur le nombre d'événements ni le moment où ils se sont produits. Les renseignements obtenus de cette façon ont une valeur marginale pour les attaques fondées sur le temps et ne peuvent qu'être utilisées pour générer des statistiques générales sur l'activité de clavier, ce qui ne constitue pas une menace dans la plupart des environnements multi-utilisateurs.

Les générateurs de nombres aléatoires matériels : une meilleure solution ?

De nos jours, un certain nombre de plates-formes matérielles implémentent des générateurs de nombres aléatoires physiques, appelés vrais générateurs de nombres aléatoires

(TRNG, de l'anglais *True Random Number Generators*). Ces périphériques offrent un moyen plus fiable de générer des données vraiment imprévisibles par rapport à la collecte d'informations dont on espère simplement qu'elles soient difficiles à prédire. Il est conseillé d'utiliser ce moyen pour acquérir de l'entropie sur toutes les machines équipées de ce matériel. Au moment de la rédaction de cet ouvrage, deux solutions fondées sur des circuits intégrés sont mises au point par Intel et VIA.

Intel intègre un générateur de nombres aléatoires dans certains de ses chipsets, comme i810. Le système utilise une conception classique de deux oscillateurs : l'oscillateur de haute fréquence génère un signal de base qui est pour l'essentiel une alternance logique d'états (010101010101...) et un oscillateur à basse fréquence, travaillant à un taux nominal de 1/100 de la fréquence de l'oscillateur à haute vitesse, mais dont la fréquence est modulée par une résistance qui sert de principale source d'entropie.

Certaines caractéristiques mesurables d'une résistance changent en raison d'un bruit thermique et d'autres effets matériels aléatoires. L'oscillateur à basse fréquence est utilisé pour conduire des échantillons du signal alternatif à des fréquences aléatoires (crête basse de la sortie de l'oscillateur). Le signal, après conditionnement et "blanchiment" *via* la correction de von Neumann, est alors mis à la disposition du monde extérieur. À travers une analyse minutieuse de la conception et de la sortie effective du générateur, Benjamin Jun et Paul Kocher, de Cryptography Research[9], ont montré que la qualité de la sortie est constamment élevée et que le générateur fournit environ 0,999 bit d'entropie par bit de sortie.

Le générateur de nombres aléatoires du processeur VIA C3 "Nehemiah" repose sur une conception légèrement différente, puisqu'il utilise un ensemble d'oscillateurs mais pas une source de bruit distincte, comme une résistance spéciale. Au lieu de cela, il s'appuie sur le mouvement interne des oscillateurs, un effet qui peut être attribué à un certain nombre de facteurs internes et externes et qui peut être contrôlé par un paramètre "bias" configurable.

Dans ce cas, une autre analyse dirigée par Cryptography Research[10] a indiqué que le générateur délivre apparemment une entropie de moins bonne qualité que son homologue, allant de 0,855 à 0,95 bit par bit en sortie. Ce résultat est dangereux si la sortie du générateur de nombres aléatoires est considérée comme entièrement aléatoire et utilisée pour la génération de clé ou d'autres tâches critiques 1:1, car le montant réel de l'entropie est réduit en conséquence. Pour résoudre ce problème, on peut collecter plus de données que nécessaire du générateur, puis exécuter les données *via* une fonction de hachage sécurisée comme SHA-1, afin d'éliminer une éventuelle tendance ou un manque d'entropie. En règle générale, cette solution représente une bonne solution pour éviter les effets indésirables des TRNG, tant que ceux-ci restent dans des limites raisonnables, autrement dit tant que chaque bit véhicule une part d'entropie utilisable.

Plusieurs chercheurs ont également suggéré d'utiliser comme source d'entropie certains périphériques d'entrée non spécialisés, par exemple des webcams ou des microphones intégrés : les capteurs CCD (*Charge Coupled Device*) des appareils photo numériques tendent à créer des pixels de bruit tandis que le signal d'un microphone produit une bonne source de bruit aléatoire lorsqu'il est fortement amplifié. Toutefois, il n'existe pas de méthode universelle pour paramétrer un de ces générateurs. En effet, les fabricants de ces dispositifs utilisent chacun des circuits différents, si bien que la qualité des nombres "aléatoires" générés de cette façon ne peut être garantie. En fait, si certains dispositifs semblent collecter des interférences radio ou certains signaux en entrée apparemment au hasard, ils le font en fait de façon totalement prévisible. En outre, certains périphériques, en particulier les capteurs CCD, produisent un bruit électronique statique. Ce bruit, qui semble aléatoire, évolue très lentement, et il peut donc être dangereux de s'y fier.

Matière à réflexion

J'ai décidé de ne pas examiner en détail un petit nombre de concepts intéressants, mais ils peuvent être une précieuse source d'inspiration pour de plus amples recherches.

Les attaques à distance fondées sur le temps

En théorie, il doit être possible de déployer une attaque fondée sur le temps d'opération du PRNG sur un réseau. Certains services de cryptographie implémentent une cryptographie symétrique. Après avoir établi un flux asymétrique à clé publique plus lent et vérifié les deux parties, une clé de session symétrique est générée et les deux terminaux passent à une alternative symétrique plus rapide.

- Il pourrait être possible de mesurer l'intervalle de temps entre chaque touche en forçant l'application à épuiser le réservoir d'entropie dans le système, au point qu'il manque une toute petite quantité d'entropie pour générer une nouvelle clé de session. L'application va alors retarder la génération d'une clé symétrique jusqu'à ce que suffisamment d'entropie soit disponible pour créer le reste de la clé, ce qui va se produire, entre autres possibilités, la prochaine fois qu'une touche sera enfoncée ou relâchée.
- Je suis convaincu que l'attaque a plus de chances de réussir dans un laboratoire que dans le monde réel, bien que mon conseiller technique soit en désaccord avec mon scepticisme. Considérez donc ceci comme mon avis personnel. Une analyse intéressante de l'université de Virginie a critiqué les recherches initiales sur le temps d'opération SSH abordées dans le document mentionné précédemment, au motif que les variations du réseau sont suffisantes pour rendre inutilisables les données sur le

temps d'opération. Pourtant, il est intéressant de noter que, si une activité spécifique est répétée dans le temps (le même mot de passe entré à chaque nouvelle connexion, par exemple), les fluctuations aléatoires des performances du réseau peuvent très bien révéler un temps moyen d'opération[11].

Exploiter les diagnostics du système

Certains systèmes disposent de meilleurs moyens pour écouter le clavier et obtenir d'autres données sur le temps d'opération. Après la publication de mon étude sur le temps d'opération des PRNG, on m'a signalé que Linux contient une interface /proc/interrupts qui affiche une synthèse statistique des interruptions, dans le but de fournir certaines données utiles sur les performances. En examinant le nombre de changements des interruptions de IRQ 1, il est possible d'obtenir les mêmes informations sur le temps d'opération qu'avec le générateur de nombres pseudo-aléatoires, déjà filtrés de toute activité du disque et de réseau éventuelle, ce qui provoque une exposition de la vie privée semblable à celle dont nous avons parlé auparavant.

Reproduction de l'imprévisible

D'autres questions liées à l'implémentation du PRNG lui-même méritent d'être examinées. Il est courant de procéder à l'achat groupé de matériel identique et à l'installation du même système sur plusieurs machines, ce qui peut se révéler un problème pour les serveurs qui n'ont pas une activité importante au niveau console. Les outils spécialisés qui permettent de créer une image d'une installation et de la propager sur plusieurs serveurs présentent également un risque. Dans tous les cas, les systèmes risquent de disposer d'une entropie peu élevée pendant un temps sans doute trop long.

2

Des efforts supplémentaires ne sont jamais inutiles

Où l'on apprend à créer un ordinateur en bois et à obtenir des informations en observant l'exécution d'un vrai ordinateur.

Les données que vous avez saisies sont à présent dans les mains de l'application que vous avez choisi d'exécuter. Ce programme prend alors son temps pour décider quoi faire de ces informations, comment les interpréter et quelles actions effectuer ensuite.

Dans ce chapitre, nous allons examiner en détail les mécanismes de bas niveau du traitement des données et étudier certains des pièges qui peuvent se cacher profondément sous le dissipateur de chaleur de votre processeur. Nous prêterons une attention particulière aux informations que nous pouvons déduire en observant simplement la façon dont une machine donnée exécute les programmes et le temps qu'il lui faut pour compléter certaines tâches. Enfin, nous allons également construire un ordinateur en bois complètement fonctionnel.

L'héritage de Boole

Pour comprendre la conception d'un processeur, il nous faut revenir à une époque où le concept de processeur n'avait même pas été imaginé. Tout a commencé assez innocemment au XIXe siècle, lorsque le mathématicien autodidacte George Boole (1815-1864) mit au point un simple système de calcul binaire destiné à permettre la compréhension et la modélisation du calcul formel. Son approche réduisit les concepts fondamentaux de la logique à un ensemble de trois opérations algébriques simples qui pouvaient être appliquées à des éléments représentant deux états opposés, vrai ou faux. Ces opérations sont les suivantes :

- L'opérateur de disjonction **OR**. Vrai lorsqu'au moins un des opérandes* est vrai**.
- L'opérateur de conjonction **AND**. Vrai uniquement lorsque tous les opérandes sont vrais.
- L'opérateur de complément (négation), **NOT**. Vrai lorsque l'opérande unique est faux.

Bien que d'une conception simple, le modèle algébrique booléen s'est révélé un puissant outil pour résoudre des problèmes de logique et certains autres défis mathématiques. Il a même permis à de nombreux visionnaires d'imaginer les astucieuses machines analytiques qui ont un jour changé notre vie quotidienne.

De nos jours, tous les utilisateurs expérimentés connaissent bien la logique booléenne, mais rares sont ceux qui savent comment cet ensemble d'opérations simples fut appliqué aux ordinateurs actuels. Nous allons commencer à explorer cette voie en tentant de saisir l'essence de ce modèle dans son expression la plus simple.

Vers l'opérateur universel

Le chemin qui mène à la simplicité emprunte souvent des étapes complexes apparemment inutiles, et notre exemple ne fait pas exception. Pour commencer, nous devons tenir compte des travaux d'un autre mathématicien du XIXe siècle, Augustus De Morgan (1806-1871). La loi de De Morgan stipule "qu'un complément de la disjonction est la conjonction de compléments". Cet exercice qui complexifie certains concepts simples eut des conséquences profondes sur la logique booléenne et, finalement, sur la conception des circuits numériques.

En clair, la loi de De Morgan explique que, si une des deux conditions (ou les deux) n'est pas remplie, une phrase qui affirme que les deux conditions sont remplies (autrement dit qu'il y a conjonction de conditions) sera également fausse. Et *vice versa*, d'ailleurs.

---

* Un opérande est un élément qui subit l'action de l'opérateur.

** Le sens du OR logique diffère du sens du mot "ou" en français. En effet, le résultat reste vrai à la fois quand un seul des paramètres OR est vrai, et lorsque tous le sont. En français, "ou" signifie généralement qu'une seule option est vraie.

Cette loi conclut que la condition NOT OR (a, b) devrait logiquement être équivalente de la condition AND (NOT a, NOT b). Prenons un exemple du monde réel dans lequel a et b sont les suivants :

```
a = "Bob aime le lait"
b = "Bob aime les pommes"
```

Les deux parties de l'équation de De Morgan peuvent être écrites ainsi :

```
OR (NOT a, NOT b) ⇔ Bob n'aime PAS le lait OU n'aime PAS les pommes
NOT AND (b, a) ⇔ Il n'est PAS vrai que Bob aime le lait ET les pommes
```

Les deux expressions sont fonctionnellement équivalentes. S'il est vrai que Bob n'aime pas le lait ou les pommes, la première expression est vraie ; il est alors également vrai qu'il n'aime pas les deux, ce qui signifie que la seconde expression est également vraie.

Inverser la situation entraîne également le même résultat : s'il n'est pas vrai que Bob n'aime pas au moins l'un des deux choix, il aime les deux à la fois (et la première expression est fausse). Dans ce cas, il n'est pas vrai non plus qu'il n'aime pas les deux à la fois (et la seconde expression est également fausse).

La loi de De Morgan en pratique

Pour évaluer les déclarations logiques au-delà de l'intuition, il est nécessaire de créer ce que l'on appelle des tables de vérité qui démontrent que tous les résultats peuvent être calculés à partir de toutes les combinaisons possibles des opérateurs true et false (vrai et faux).

Les deux tableaux suivants représentent chaque expression de l'exemple précédent. Chaque tableau contient des colonnes pour les deux opérateurs et les résultats correspondants, et ce pour toutes les combinaisons possibles (vrai et faux). Vous pouvez donc voir dans la première ligne que les deux premières colonnes – les deux opérateurs de NOT AND(a et b) – sont fausses. En conséquence, AND(a et b) est faux, et donc NOT AND(a et b) est vrai. Le résultat est noté dans la troisième colonne.

Comme vous pouvez le voir, les deux expressions se comportent de la même manière :

NOT AND(a, b) : résultat inverse de AND

Opérande 1 (a)	*Opérande 2 (b)*	*Résultat*
FALSE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

OR(NOT a, NOT b) : opérande inverse de OR

Opérande 1	*Opérande 2*	*Résultat*
FALSE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

Mais pourquoi donc les concepteurs d'ordinateur se soucient-ils des préférences alimentaires de Bob ? Parce que, dans le contexte des opérateurs booléens, la loi de De Morgan signifie que les opérations de base proposées par l'algèbre de Boole sont en fait partiellement redondantes : NOT combiné à l'un des deux autres opérateurs (OR et AND) suffit toujours pour faire la synthèse des possibilités. Par exemple :

```
OR (a, b) ⇔ NOT AND (NOT a, NOT b)
AND (a, b) ⇔ NOT OR(NOT a, NOT b)
```

En comprenant cela, on réduit le nombre d'opérateurs à deux. Mais le système booléen peut encore être simplifié.

La commodité est une nécessité

Plusieurs autres opérateurs ne sont pas indispensables pour l'implémentation de la logique booléenne mais complètent l'ensemble des opérations existantes. Ces opérateurs supplémentaires, NAND et NOR, sont vrais uniquement lorsque AND et OR sont respectivement faux :

```
NAND (a,b) ⇔ NOT AND (a,b) ⇔ OR (NOT a, NOT b)
NOR(a, b) ⇔ NOT OR (a,b) ⇔ AND (NOT a, NOT b)
```

Ces nouvelles fonctions ne sont pas plus complexes que AND et OR. Chacune a une table de vérité à quatre états (quatre lignes), et sa valeur peut être définie aussi facilement.

NOTE

NOR et NAND ne font pas partie des opérandes de base car ni l'un ni l'autre ne correspondent à un type simple de relations logiques entre des déclarations et n'ont pas d'équivalents dans le langage courant.

Je viens d'ajouter une série de nouveaux opérateurs, issus de l'ensemble existant, qui semblent n'offrir qu'une fonctionnalité plus ou moins commode à ceux qui cherchent à formuler des dépendances logiques plus bizarres ou des problèmes en utilisant la notation formelle. Dans quel but ?

À eux seuls, NAND ou NOR permettent de complètement se débarrasser de AND, OR et NOT. Ce qui correspond à notre objectif de simplicité et nous permet de décrire l'ensemble du système d'algèbre de Boole avec moins d'éléments et moins d'opérateurs.

L'importance de ces opérateurs auxiliaires de négation est que vous pouvez utiliser n'importe lequel d'entre eux pour construire un système algébrique de Boole complet. En fait, vous pouvez construire tous les opérateurs simples en utilisant NAND, comme indiqué ici (T indique une déclaration vraie et F, une déclaration fausse*). Comment ? Eh bien, bien évidemment, les paires suivantes de déclarations sont équivalentes :

```
NOT a ⇔ NAND(T, a)
AND(a,b) ⇔ NOT NAND(a,b) ⇔ NAND(T, NAND(a,b))
OR(a,b) ⇔ NAND(NOT a, NOT b) ⇔ NAND(NAND(T,a), NAND(T,b))
```

Ou, si nous préférons n'utiliser que NOR plutôt que NAND, nous pouvons écrire :

```
NOT a ⇔ NOR(F, a)
OR(a,b) ⇔ NOT NOR(a, b) ⇔ NOR(F, NOR(a,b))
AND(a,b) ⇔ NOR(NOT a, NOT b) ⇔ NOR(NOR(F,a), NOR(F,b))
```

Englober la complexité

Il peut être difficile de croire que tous les calculs peuvent être résumés à la logique de ces opérateurs universels. Vous pouvez implémenter les algorithmes les plus complexes, les calculs les plus avancés, les jeux les plus récents et la navigation sur Internet en utilisant un ensemble de circuits simples à partir des tables de vérité suivantes, qui convertissent les signaux d'entrée en signaux de sortie.

Table d'état NAND

Opérande 1	*Opérande 2*	*Résultat*
FALSE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

* Pour les puristes, T est équivalent à AND (a,a), par exemple, ce qui est toujours vrai, et que F est équivalent à NOT AND (a,a), qui est toujours faux. En d'autres termes, nous n'ajoutons ici aucun nouveau concept ou aucun nouvel élément dans l'équation, nous simplifions seulement un peu la notation.

Table d'état NOR

Opérande 1	*Opérande 2*	*Résultat*
FALSE	FALSE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	FALSE

Il semble pourtant que nous n'allons nulle part… Comment cet ensemble de dépendances simples permet-il de construire un dispositif capable de résoudre des problèmes complexes, comme le rejet de votre demande de crédit, avec tact ? Et qu'est-ce que la théorie fondée sur les états "vrais" et "faux" a de commun avec les circuits numériques ?

Vers le monde matériel

Le mécanisme conçu par Boole n'a rien de complexe : il fait appel à deux états logiques opposés, "vrai" et "faux", 0 et 1, "Cyan" et "violet", 999 et 999 ½. Le sens réel, la représentation physique, et le support ne sont pas importants ; seule compte la convention arbitrairement choisie, qui attribue certains états du support physique à un ensemble spécifique de valeurs logiques.

Les ordinateurs tels que nous les connaissons utilisent deux niveaux différents de tension dans un circuit électronique et les interprètent comme des valeurs que leurs concepteurs désignent comme étant égales à 0 et à 1. Ces valeurs, qui sont transmises à travers le circuit électrique, représentent deux chiffres dans le système binaire. Mais n'importe quelle méthode de transmission des données peut être utilisée, qu'il s'agisse de l'écoulement de l'eau, d'une réaction chimique, de signaux de fumée ou d'une série de roues artisanales en bois qui pivotent. L'information reste la même, indépendamment de son véhicule.

L'application de la logique booléenne dans le monde physique est simple, une fois que nous sommes d'accord sur la représentation physique des valeurs logiques. Il suffit ensuite de trouver un moyen d'organiser une série d'éléments pour manipuler ces valeurs afin de tenir compte de toutes les tâches que nous voulons que notre ordinateur effectue (nous reviendrons sur ce point plus tard). Tout d'abord, essayons de savoir comment manipuler des signaux et implémenter des dispositifs logiques dans le monde réel, autrement dit des portes en bois.

Un ordinateur sans électricité

Il semble compliqué de passer d'une série d'opérations théoriques purement mathématiques à un dispositif qui permette de contrôler l'écoulement de l'eau, le couple d'une transmission ou les signaux électriques d'une manière qui imite l'un des opérateurs logiques, mais ce n'est pas le cas.

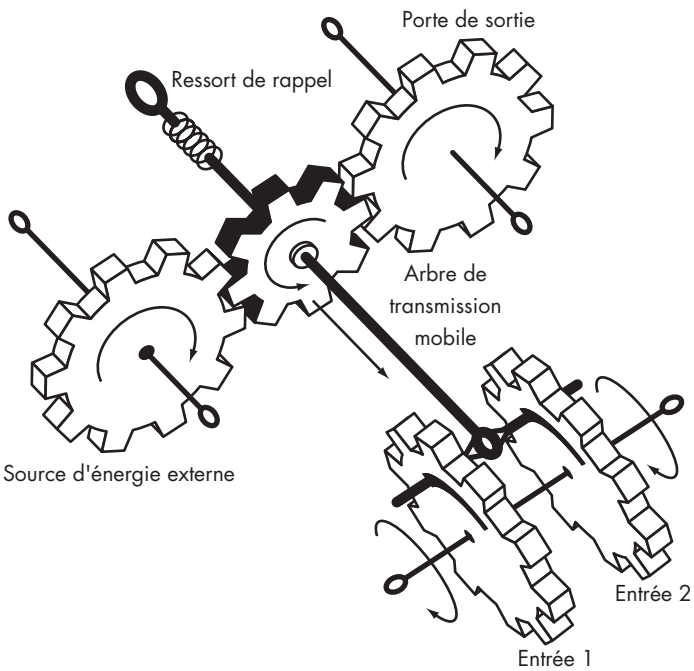


Figure 2.1
Conception d'une porte mécanique NOR.

La Figure 2.1 montre un mécanisme simple qui met en œuvre la fonctionnalité NOR en utilisant des roues dentées. À l'arrêt, la roue en "sortie" représente l'état 0 ; quand elle tourne, son état est 1. Le dispositif transmet le couple d'une source extérieure à la sortie uniquement si aucune des deux roues de contrôle en "entrée" ne tourne. En théorie, une source d'énergie extérieure n'est pas nécessaire, et le mécanisme pourrait être plus simple. Dans la pratique, toutefois, il serait assez difficile de construire un ensemble plus complexe de portes autonomes en raison notamment du frottement.

Lorsqu'une des roues en entrée (ou les deux) tourne, l'arbre de transmission se déplace et rend inactif le système en "sortie". Lorsque les entrées deviennent inactives, un ressort

de rappel replace l'arbre de transmission dans sa position initiale. La table de vérité pour ce dispositif est exactement ce que devrait être NOR.

Comme vous vous en souvenez, NOR ou NAND suffisent pour mettre en œuvre n'importe quel opérateur booléen logique. Même si l'ajout de la capacité à mettre en œuvre d'autres opérateurs sans recombiner les portes NAND et NOR rendrait notre dispositif plus petit et plus efficace, le dispositif n'a pas besoin de cette capacité pour fonctionner.

En supposant que nous parvenions à faire collaborer toutes les portes d'une manière à laquelle nous sommes habitués, nous pouvons conclure que les ordinateurs peuvent être construits avec quasiment n'importe quelle technologie*.

Une conception d'ordinateur légèrement plus classique

Même si l'essor de l'informatique des dernières décennies a été possible grâce au transistor, son importance n'a rien de magique ou ne découle pas de ses qualités uniques. Il s'agit tout simplement de l'élément le moins cher et le plus efficace dont nous disposons à l'heure actuelle.

Contrairement aux machines en bois, les signaux électriques sont relayés dans les ordinateurs électroniques grâce à des transistors, de petits dispositifs qui laissent passer le courant dans un sens entre deux de leurs nœuds (points de connexion) quand une tension est appliquée au troisième nœud. Les transistors peuvent être miniaturisés de façon tout à fait efficace, nécessitent peu d'énergie, sont fiables et bon marché.

Portes logiques

Le transistor est simple. En fait, il constitue à lui seul un dispositif trop simple pour implémenter une logique booléenne significative. Cependant, en les disposant correctement en portes logiques, les transistors permettent d'effectuer toutes les opérations de l'algèbre de Boole.

La porte AND peut être implémentée en disposant deux transistors en série, de sorte que les deux doivent avoir une faible résistance (soient en position "on") avant que la tension ne puisse être transmise vers la sortie. Chaque transistor est contrôlé (activé) par une ligne en entrée distincte. La sortie est nominalement "abaissée" en utilisant une résistance, de sorte qu'elle a une tension de départ de 0 ("faux"), mais cette tension dépassera 0 une fois que les transistors sont sur "on" et permettent à un léger courant de passer.

---

* Il est évident que les ordinateurs non électriques ne sont pas légion. On peut tout de même citer la *machine à différences* de Charles Babbage, ainsi que les nanotechnologies, qui semblent également prometteuses. Voir Ralph C. Merkle, "Two Types of Mechanical Reversible Logic", Nanotechnology 4 (1993).

La porte OR est implémentée par la mise en place d'un transistor en parallèle, de sorte que n'importe quel transistor puisse l'activer et permettre à la sortie d'avoir une tension différente de zéro, autrement dit "vrai".

La dernière porte de base, NOT, est mise en place en utilisant un seul transistor et une résistance. Son état en sortie est égal à 1 lorsqu'elle est inactive (grâce à la résistance) et est abaissé à 0 lorsque le transistor s'ouvre (voir Figure 2.2).

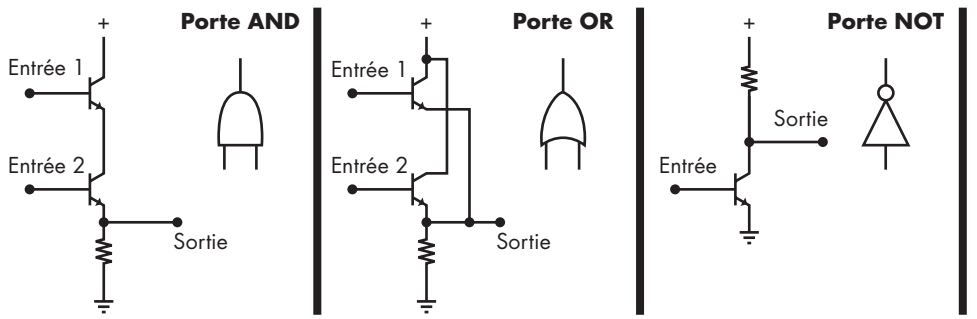


Figure 2.2

Portes logiques utilisant des transistors – construction et symboles.

NOTE

Vous avez peut-être remarqué que les deux portes AND et OR peuvent être transformées en portes NAND et NOR sans introduire de composants supplémentaires. Il suffit d'utiliser une conception basée sur l'observation des schémas d'une porte NOT, autrement dit de déplacer la résistance et le "point de sortie" vers la tension d'alimentation et donc d'inverser la sortie logique.

Nous avons maintenant atteint un point où nous pouvons combiner les transistors pour implémenter une des portes universelles. Mais, quel que soit le nombre de portes que nous pouvons construire, le système reste encore assez éloigné d'un vrai ordinateur.

Tout ce que nous venons de voir est intéressant, mais en quoi la logique booléenne peut nous aider à autre chose que résoudre des énigmes sur l'alimentation de Bob ?

Des opérateurs logiques aux calculs

En combinant des opérations booléennes logiques simples, on peut aboutir à certains résultats étonnants, notamment la possibilité d'effectuer des opérations arithmétiques sur la représentation binaire des nombres. C'est là que les choses deviennent intéressantes.

Un ensemble de portes XOR et AND, par exemple, peut être utilisé pour augmenter la valeur d'une entrée de 1, ce qui constitue la première étape vers la mise en place d'une addition. La Figure 2-3 illustre la création d'un compteur basé sur ce concept.

Ah, une nouvelle expression ! XOR est encore un opérateur logique booléen "pratique" qui est vrai uniquement lorsque l'un de ses opérandes est vrai. À cet égard, il est plus proche du sens habituel de "ou" en français. XOR est souvent utilisé pour simplifier la notation, mais également facile à mettre en œuvre par d'autres moyens, en combinant AND, NOT et OR. Il est défini de la manière suivante :

```
XOR(a, b) ⇔ AND(OR(a,b), NOT AND(a,b))
```

Revenons à notre circuit… Que peut-il faire ? Le dispositif représenté à la Figure 2.3 est alimenté par un nombre écrit en binaire. Dans cet exemple, ce nombre est limité à 3 bits, bien que ce modèle puisse facilement être amélioré pour permettre un plus grand nombre d'entrées.

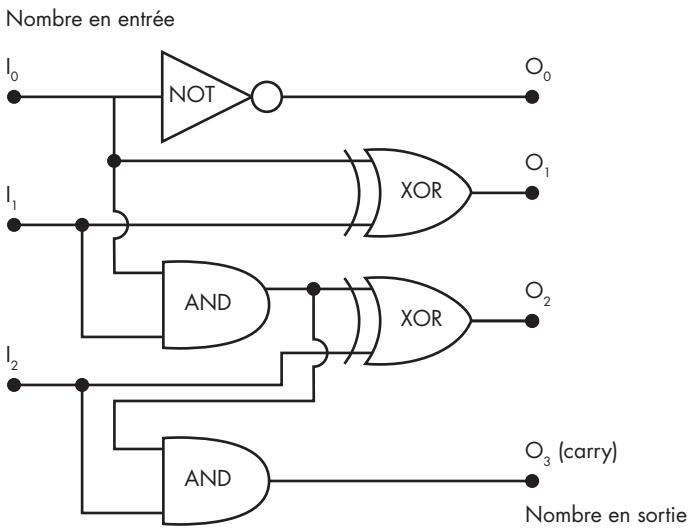


Figure 2.3

Circuit simple incrémentant la valeur de 1.

Cette simple unité de calcul fonctionne de la même façon que les êtres humains lorsqu'ils ajoutent des nombres décimaux sur une feuille de papier – de la droite vers la gauche, en reportant éventuellement une valeur dans la colonne suivante. La seule véritable différence réside dans le fait qu'elle utilise un système binaire.

Étudions son fonctionnement. Nous avons un nombre binaire écrit sur une ligne. Nous voulons augmenter sa valeur de 1 ; nous commençons par le chiffre le plus à droite, de la même façon que nous le ferions dans une addition de décimales.

Nous disposons d'un chiffre binaire ; en augmentant la valeur d'un chiffre binaire de 1, seuls deux résultats sont possibles : si le chiffre en entrée est 0, le résultat est 1 (0 + 1 = 1) ; dans le cas contraire, la sortie est 0, et nous devons reporter la valeur 1 dans la colonne suivante (1 + 1 = 10). En d'autres termes, nous faisons deux choses : nous produisons un produit qui est une négation de l'entrée (1 pour 0 et 0 pour 1) et, si le chiffre en entrée est 1, nous devons le garder à l'esprit et l'inclure plus tard.Or c'est justement ce que réalise le circuit : pour la première entrée, I_0 (Input 0). La première porte d'entrée traite l'entrée en créant son négatif par la négation, transmet cette valeur à la sortie O_0 (Output 0) et alimente avec cette valeur en entrée les autres portes qui ont la charge de gérer la colonne suivante (O_1).

O_0 = NOT I_0
C_0 = I_0

Bon, nous avons augmenté le nombre de 1. Il n'y a rien d'autre à faire dans les colonnes restantes si la valeur n'est pas transmise par la colonne précédente. Si rien n'est transmis, O_1 devrait être égal à I_1. Si une valeur est transmise, en revanche, nous devons alors traiter cette valeur de la même façon que nous l'avons fait en ajoutant 1 à la colonne précédente, inverser la sortie et transmettre une valeur à la colonne suivante s'il y a lieu.

À partir de maintenant, chaque sortie (O_n avec $n > 0$) sera soit copiée directement à partir de I_n si aucun bit n'est reporté de la colonne précédente soit augmentée de 1 (ce qui, encore une fois, revient à inverser la valeur) en raison de l'addition de la valeur du bit transmis. Donc, si I_n est égal à 1, le report de cette colonne, C_n, sera également égal à 1 et O_n sera égal à 0 (puisqu'en binaire 1 + 1 a pour résultat 10). Comme vous l'avez peut-être remarqué, la sortie réelle à la position *n* est simplement un résultat XOR de la valeur d'entrée à la position *n* et du bit en provenance de la colonne *n – 1*. Ainsi, le circuit génère O_n en effectuant une opération XOR sur le bit transmis par C_{n-1} et la valeur de I_n, puis effectue une opération AND sur la valeur transmise depuis O_{n-1} et I_n afin de déterminer s'il devrait y avoir un report de la valeur dans la colonne suivante :

O_n = XOR(I_n,C_{n-1})
C_n = AND (I_n,C_{n-1})

Prenons l'exemple suivant. Nous voulons augmenter la valeur d'entrée 3 (011 en binaire) de 1. Les entrées sont les suivantes :

I_0 = 1
I_1 = 1
I_2 = 0

Le circuit produit O_0 par négation de I_0 et donc $O_0 = 0$. Comme I_0 était différent de zéro, il est également reporté dans la colonne suivante. Dans la colonne suivante, la porte XOR donne à O_1 une valeur de 0 car, même si I_1 était égal à 1, il y avait une valeur

transmise depuis la colonne précédente (1 + 1 = 10). Là encore, la valeur est reportée dans la colonne suivante.

Dans une autre colonne, $I_2 = 0$, mais la porte AND indique une valeur reportée de la rangée précédente, car deux entrées précédentes ont toutes deux été fixées à 1. Par conséquent, la sortie est 1. Il n'y aura aucun report dans la dernière colonne. La sortie est la suivante :

$O_0 = 0$
$O_1 = 0$
$O_2 = 1$
$O_0 = 0$

…ou 0100, ce qui, accessoirement, est égal à 4, une fois converti en nombres décimaux. Et voilà, nous avons écrit +1 en binaire.

NOTE

Nous venons d'exprimer le premier problème informatique en termes algébriques booléens. Vous pourriez être tenté d'étendre la conception afin qu'elle puisse additionner deux numéros arbitraires, et pas simplement un nombre et le nombre 1. Néanmoins, ce circuit représente une base suffisante pour tous les calculs.

Les circuits arithmétiques numériques fonctionnent en exécutant certaines données d'entrée à travers une série de portes logiques habilement disposées qui, à leur tour, ajoutent, soustraient, multiplient ou effectuent des modifications simples sur un tableau de bits. Rien de magique, donc.

Jusqu'à présent, j'ai expliqué comment les puces de silicium ou les mécanismes artisanaux en bois effectuent certaines opérations simples, comme le calcul sur les entiers. Pourtant, quelque chose manque. En effet, les éditeurs de texte, les jeux et les logiciels de peer to peer ne sont pas codés en dur dans un système complexe de portes à l'intérieur du processeur. Qu'en est-il des logiciels ?

Du sablier électronique à l'ordinateur

Le véritable intérêt d'un ordinateur réside dans sa capacité à être programmé pour agir d'une manière spécifique, autrement dit pour exécuter une suite de commandes logicielles prédéfinies.

La Figure 2.4 illustre la prochaine étape sur le chemin vers le développement d'une machine flexible qui puisse effectuer plus qu'une seule tâche basée sur des fils métalliques, à savoir le stockage de données et la mise en mémoire. Dans cette figure, nous voyons un type d'unité de stockage de mémoire connue sous le nom de *conception*

flip-flop (bistable). Cette case mémoire compte deux lignes de contrôle, "set" et "reset". Lorsque les deux sont désactivées, la porte conserve son état actuel, grâce à un lien entre son entrée et la sortie de la porte OR. La sortie précédente de l'opération OR est transmise à travers une porte AND parce que sa ligne est définie à 1 (négation "reset"), puis de nouveau à travers OR parce que son autre entrée est 0 ("set"). L'état de la sortie est maintenu aussi longtemps que les portes sont alimentées.

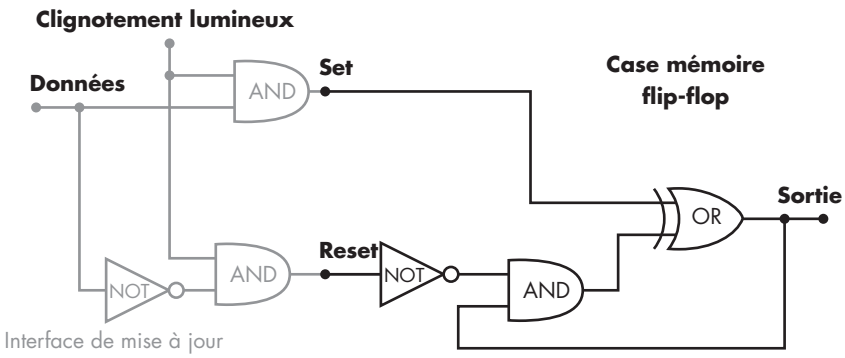


Figure 2.4

Mémoire flip-flop avec une interface en pratique.

Lorsque la valeur "set" augmente, la porte OR est forcée de renvoyer 1 et conserve cette valeur quand "set" diminue. Lorsque la ligne "reset" augmente, la porte AND est forcée de renvoyer 0 et brise la boucle de rétroaction, ce qui oblige le circuit à produire 0 en sortie. Une fois que "reset" diminue, la sortie reste égale à 0. Lorsque les deux lignes de contrôle sont activées, le circuit devient instable – quelque chose qui n'est pas très joli, surtout lorsque l'ordinateur en question est mécanique.

La table de vérité pour ce modèle se présente comme suit (V indique une valeur logique arbitraire).

Table de vérité flip-flop

Set	*Reset*	Q_{t-1}	Q_t
0	0	*V*	*V*
1	0	-	1
0	1	-	0
1	1	-	instable

Une version plus pratique d'un circuit flip-flop qui intègre une "interface de mise à jour"(voir Figure 2.4) fait appel à deux portes AND et à une porte NOT afin que l'état d'une ligne d'entrée soit capturé (échantillonné et maintenu) chaque fois qu'un clignotement lumineux externe se produit. Cette conception élimine les combinaisons instables d'entrées et rend cette sorte de mémoire plus facile à utiliser pour stocker de l'information.

Table de vérité flip-flop améliorée

Entrée	*Clignotement lumineux*	Q_{t-1}	Q_t
–	0	V	V
S	1	–	S

Cette configuration simple possède une propriété importante : elle peut stocker des données. Une seule case mémoire ne peut stocker qu'un bit mais, en combinant un certain nombre de flip-flops, on peut augmenter cette capacité de stockage. Même si la mémoire est conçue différemment de nos jours, l'importance de cette fonctionnalité reste la même : elle permet aux programmes de s'exécuter. Mais comment ?

Dans cette conception de base, la puce stocke une valeur spéciale, généralement appelée le pointeur d'instruction, dans une mémoire intégrée à la puce (le registre), composée de plusieurs flip-flops. Comme les ordinateurs fonctionnent de façon synchrone et que tous les processus sont gérés par un signal d'horloge à haute fréquence, le pointeur d'instruction sélectionne une case mémoire de la mémoire principale à chaque cycle d'horloge. Les données de contrôle extraites de cette façon activent puis sélectionnent le circuit de calcul approprié pour traiter les données d'entrée.

Pour certaines données de contrôle, notre puce hypothétique effectue une addition ; pour d'autres, elle effectue une opération d'entrée/sortie. Après avoir collecté chaque partie des données de contrôle (chaque instruction de la machine), la puce prépare son pointeur d'instruction interne à lire la prochaine commande dans le cycle suivant. Grâce à cette fonctionnalité, on peut utiliser la puce pour exécuter une suite d'instructions machine ou un programme.

Il est maintenant temps de savoir quelles opérations la puce doit implémenter pour être utilisable.

La machine de Turing et les suites complexes d'instructions

En fait, le processeur n'a pas besoin d'être complexe. Le jeu d'instructions nécessaires pour qu'une puce soit capable d'exécuter à peu près n'importe quelle tâche est étonnamment faible. La thèse de Church et Turing affirme que tout calcul réalisable dans le monde réel peut être effectué par une machine de Turing, qui est un modèle primitif d'ordinateur. La machine de Turing, du nom de son inventeur, est un dispositif simple qui opère sur un ruban potentiellement infini divisé en cases consécutives, qui sont chacune un emplacement de stockage purement hypothétique et abstrait. Chaque case peut stocker un caractère unique à partir d'un "alphabet", autrement dit un ensemble fini et ordonné de valeurs possibles (cet alphabet n'a absolument rien à voir avec les alphabets des langues humaines ; son nom ne sert qu'à créer une bonne dose de confusion).

L'appareil est également équipé d'un registre interne qui mémorise un nombre fini d'états internes. Au début de son exécution, la machine de Turing commence à une certaine position sur le ruban, dans un état donné, puis lit un caractère d'une case sur le ruban. Chaque automatisme est associé à un ensemble de motifs de transitions qui indiquent à la machine comment modifier l'état interne de la machine, quel symbole de l'alphabet doit être stocké sur le ruban en fonction de l'état de la machine après lecture, et comment (en option) déplacer la tête de lecture d'une case vers la gauche ou la droite. Cette suite de transitions définit les règles pour le calcul du prochain état du système en fonction de ses caractéristiques courantes. Ces règles sont souvent documentées au moyen d'une table d'actions du genre suivant.

Table d'actions

État courant		***Nouvel état/action***		
C_t	S_t	C_{t+1}	S_{t+1}	Mouvement
0	S0	1	S1	–
1	S0	0	S0	Gauche

La table nous indique que, si la valeur courante de la case sur laquelle la machine est positionnée est égale à 0 et que l'état interne de la machine à ce moment soit S0, le dispositif modifiera l'état de C à 1, modifiera son état interne à S1 et ne déplacera pas la tête de lecture.

La Figure 2-5 montre un exemple d'une machine de Turing placée sur la case C avec un état interne S.

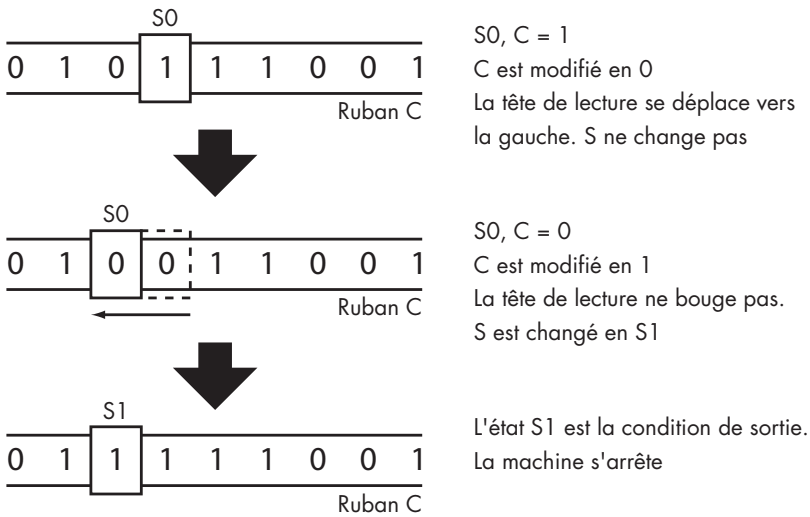

S0
0 1 0 1 1 1 0 0 1
Ruban C
S0, C = 1
C est modifié en 0
La tête de lecture se déplace vers
la gauche. S ne change pas
S0
0 1 0 0 1 1 0 0 1
Ruban C
S0, C = 0
C est modifié en 1
La tête de lecture ne bouge pas.
S est changé en S1
S1
0 1 1 1 1 1 0 0 1
Ruban C
L'état S1 est la condition de sortie.
La machine s'arrête

Figure 2.5

Exemple des étapes d'exécution de la machine de Turing.

Étudions d'un peu plus près ce fonctionnement. Comme vous pouvez le voir à la Figure 2.5, la machine utilise un alphabet de deux caractères, 0 et 1, et a deux états internes, S0 et S1. Elle commence avec S0 (les conditions initiales peuvent être définies arbitrairement ; j'ai choisi de commencer avec cette valeur sans aucune raison particulière). Lorsqu'elle est placée à la fin (le bit le moins significatif) d'un nombre binaire stocké sur le ruban (C_0), la machine suit la logique suivante :

- Si le caractère sous la tête de lecture de la machine est 0, il devient 1 et l'état de la machine devient S1, selon la première règle de transition documentée dans le tableau précédent. Comme il n'existe pas de règle de transition de S1, la machine s'arrête au cycle suivant.
- Si le caractère sous la tête de lecture est 1, il est changé en 0, et l'état reste le même. La machine déplace également la tête de lecture sur le ruban vers la gauche, en fonction de la deuxième règle de transition. L'ensemble du processus se répète ensuite à partir de ce nouvel emplacement, car la machine conserve son état actuel, pour lequel des règles de transition sont définies.

Utilisation pratique, enfin

Bien que cela puisse surprendre, cette machine est réellement utile et a plus qu'une valeur théorique car elle effectue des calculs de base. Elle fait exactement la même chose que le circuit qui augmente une valeur de un en un dont nous avons parlé plus tôt dans ce chapitre. En fait, elle applique le même algorithme ; les bits du ruban, en commençant par ceux situés les plus à droite, sont inversés jusqu'à atteindre la valeur 0 (qui est aussi inversée).

Il ne s'agit là naturellement que de la partie visible de l'iceberg. Une bonne machine de Turing peut implémenter tous les algorithmes jamais conçus. Le seul problème est que chaque algorithme requiert un ensemble distinct de règles de transitions et d'états internes. En d'autres termes, une nouvelle machine de Turing doit être construite pour chaque nouvelle tâche, ce qui n'est pas très pratique à long terme.

Heureusement, une forme spéciale de cette machine, la machine de Turing universelle (UTM de l'anglais *Universal Turing Machine*) dispose d'un jeu d'instructions suffisamment avancé pour implémenter toutes les machines de Turing et exécuter n'importe quel algorithme, sans qu'il soit nécessaire de modifier le tableau de transition.

Cette supermachine n'est ni particulièrement complexe ni totalement abstraite, puisqu'une machine de Turing peut être conçue pour effectuer tous les algorithmes finis (conformément à la thèse de Church Turing précédemment mentionnée). Comme la méthode "d'exécution" d'une machine de Turing est en soi un algorithme fini, une machine peut être mise au point pour l'exécuter.

Quant à la complexité de cette machine, une machine dont l'alphabet contient deux éléments ou un bit (soit l'UTM la plus petite) nécessite vingt-deux états internes et instructions pour décrire les transitions d'état et exécuter des algorithmes sur un ruban de mémoire séquentiel et infini[1]. Ce qui n'est pas si compliqué.

Le Graal : l'ordinateur programmable

La machine de Turing est également beaucoup plus qu'un dispositif hypothétique et abstrait que les mathématiciens utilisent pour se divertir. Il s'agit d'une construction qui demande à être mise en œuvre à partir d'un dispositif électronique (ou mécanique) fondé sur la logique booléenne. Peut-être même que ce dispositif peut être amélioré pour devenir beaucoup plus utile et se rapprocher un peu plus de l'informatique fonctionnel. Le seul problème vient du fait que la condition *sine qua non*, un ruban en entrée d'une longueur infinie, ne peut pas être remplie dans le monde réel. On pourrait néanmoins utiliser une grande quantité de rubans pour rendre une machine de Turing réelle assez utilisable pour répondre à la plupart des problèmes quotidiens. Et voici l'ordinateur universel.

Les ordinateurs tels que nous les connaissons ne se contentent bien sûr pas d'accéder de façon séquentielle à la mémoire, ce qui réduit sensiblement le jeu d'instructions nécessaires pour atteindre le même degré d'exhaustivité que la machine de Turing. Une machine de Turing universelle avec un alphabet de dix-huit caractères ne nécessite que deux états internes pour fonctionner. Les ordinateurs, en revanche, opèrent généralement sur un "alphabet" d'au moins 4 294 967 296 caractères (32 bits) et souvent beaucoup plus, ce qui permet des accès à la mémoire non séquentiels, l'utilisation d'un grand nombre de registres et un nombre astronomique d'états internes possibles.

En fin de compte, le modèle de la machine de Turing universelle et la pratique quotidienne confirment qu'il est possible de construire une unité de traitement flexible et programmable qui n'utilise qu'une poignée de fonctionnalités, composée de deux ou trois registres internes (pointeur d'instruction, tête de lecture/écriture des données, et peut-être accumulateur) et d'un petit ensemble d'instructions. Il est parfaitement possible d'assembler un tel dispositif avec seulement quelques centaines de portes logiques, même si les modèles d'aujourd'hui peuvent utiliser beaucoup plus.

Comme vous pouvez le voir, l'idée de construire un ordinateur à partir de zéro, même s'il est en bois, n'est pas si absurde.

Des améliorations par simplification

Un tel ensemble d'instructions ne va bien entendu pas rendre le dispositif rapide ou facile à programmer. Les machines de Turing universelles peuvent à peu près tout faire (en raison principalement de leur simplicité), mais elles sont désespérément lentes et difficiles à programmer, à tel point qu'implémenter un système de traduction assistée par la machine pour convertir plusieurs langues humaines en code machine est difficile, du moins sans que le programmeur ne devienne fou.

Les architectures ou les langages de programmation qui se rapprochent trop du modèle exhaustif de Turing (Turing-complet) sont souvent appelés *Turing tarpits* (fourre-tout de Turing). Cela signifie que, s'il est théoriquement possible de l'utiliser pour réaliser à peu près n'importe quelle tâche, cela est dans la pratique à peine possible, demande trop de temps et est trop lourd à mettre en place pour qu'il soit véritablement intéressant d'essayer. Même des tâches aussi simples que la multiplication d'entiers ou le déplacement du contenu de la mémoire peuvent nécessiter une éternité à mettre en place, et encore plus de temps pour s'exécuter. Moins l'ordinateur aura d'efforts à produire et demandera de temps pour remplir des tâches simples et répétitives, et plus le nombre de tâches qui doivent être effectuées en utilisant un nombre d'instructions distinctes sera réduit, mieux cela sera.

Un moyen populaire d'améliorer les fonctionnalités et les performances d'une unité de traitement consiste à implémenter certaines tâches communes dans le matériel qui seraient assez gênantes à effectuer dans les logiciels. Ces tâches sont mises en place en utilisant un ensemble de circuits spécialisés (notamment la multiplication et le processus de rejet d'un prêt immobilier), ce qui ajoute des extensions pratiques à l'architecture et permet un déploiement plus rapide et plus sain de programmes, tout en permettant au système d'exécuter ces fonctions dans un ordre programmé et flexible.

Curieusement, au-delà des quelques mesures initiales, il n'est pas toujours souhaitable lors de la conception d'un processeur d'augmenter de façon linéaire la complexité des circuits pour qu'il atteigne des vitesses plus élevées, utilise l'énergie plus efficacement et fournisse un ensemble amélioré de fonctionnalités. Vous pouvez bien sûr construire un grand nombre de circuits pour gérer à peu près n'importe quelle opération complexe récurrente que vous pouvez envisager. Cependant, cela ne sera pas utilisable tant que l'architecture ne sera pas véritablement mature et que votre budget ne vous aura pas permis d'investir des ressources supplémentaires dans la construction de la puce. Sur une telle plate-forme, les programmes nécessitent en effet moins de temps à s'exécuter et sont plus faciles à écrire, mais l'appareil est beaucoup plus difficile à construire, demande plus de puissance et peut devenir trop volumineux ou trop coûteux pour une utilisation routinière. Les algorithmes complexes de divisions ou de calculs à virgule flottante nécessitent un nombre extrêmement important de portes généralement inactives pour exécuter ce genre de tâche en une seule étape.

Le partage des tâches

Plutôt que suivre naïvement ce chemin coûteux et construire des blocs à même de réaliser des instructions entières en une fois, il est préférable de renoncer au modèle d'exécution en un seul cycle de l'exécution jusqu'à ce que vous disposiez d'un concept de travail et de beaucoup de temps pour l'améliorer. La meilleure méthode pour que le matériel réalise des fonctionnalités complexes consiste à morceler le travail en petits morceaux et à exécuter les tâches complexes en un certain nombre de cycles.

Dans une conception multicycle de la sorte, le processeur passe par un certain nombre de phases internes, comme dans l'exemple de la machine de Turing. Il gère les données par le biais de circuits simples dans le bon ordre et implémente donc une fonctionnalité plus complexe étape par étape, en s'appuyant sur plusieurs composants de base. Plutôt qu'utiliser un dispositif complexe qui effectue tous les calculs à la fois, il utilise par exemple un circuit pour multiplier les suites de bits des entiers en 32 bits, effectue le suivi des valeurs transmises et procède ensuite au résultat final au cours du 33^{e} cycle.

Il peut également procéder indépendamment à certaines tâches en prévision de l'opération suivante. Cela nous libère de l'obligation de concevoir des dizaines de circuits pour chaque variante d'un code opératoire, selon l'endroit depuis lequel il doit obtenir les opérandes ou stocker les résultats.

Cette approche présente également l'avantage de rendre plus efficace la gestion des ressources matérielles : pour les opérandes simples, un algorithme dont la complexité est variable peut se terminer plus tôt, en ne prenant que le nombre de cycles absolument nécessaire. Par exemple, une division par 1 risque fort de nécessiter moins de temps qu'une division par 187 371.

Un circuit simple, peu coûteux, utilisé très fréquemment et dont le temps d'exécution est variable peut représenter une solution plus rentable en terme de coût qu'un circuit complexe, consommateur de beaucoup d'énergie et dont le temps d'exécution est constant. Bien que certains des processeurs d'aujourd'hui tentent d'utiliser de plus en plus un nombre fixe de cycles pour mener à terme les tâches, presque tous ont pour ancêtres des architectures multicycles. Et vous verrez que même eux fonctionnent rarement en un seul cycle.

Mais, tout d'abord, examinons comme le net avantage que procure la simplicité d'exécution en plusieurs cycles peut avoir des effets indésirables.

Étapes d'exécution

Une des variantes de l'exécution en plusieurs cycles consiste à diviser une tâche non pas en une suite de tâches répétitives mais plutôt en un certain nombre d'étapes de préparation et d'exécution. Cette méthode, appelée *staging*, est aujourd'hui utilisée par les processeurs actuels pour améliorer leurs performances sans pour autant devenir nécessairement plus complexes. Le staging de l'exécution est devenu l'une des plus importantes fonctionnalités des processeurs.

Aujourd'hui, les processeurs peuvent traduire chaque instruction en une série de petites étapes largement indépendantes. Certaines étapes peuvent être réalisées en utilisant des circuits génériques partagés par toutes les instructions, ce qui contribue à la simplicité de l'ensemble. Par exemple, un circuit dédié à une tâche donnée (la multiplication vient à l'esprit une fois de plus) peut être rendu plus universel et réutilisable et prendre part à des instructions plus complexes en lui évitant toute tâche de gestion des entrées-sorties génériques, et ainsi de suite. Les étapes d'exécutions et de transitions dépendent de l'architecture, mais le principe est généralement semblable à celui indiqué à la Figure 2.6.

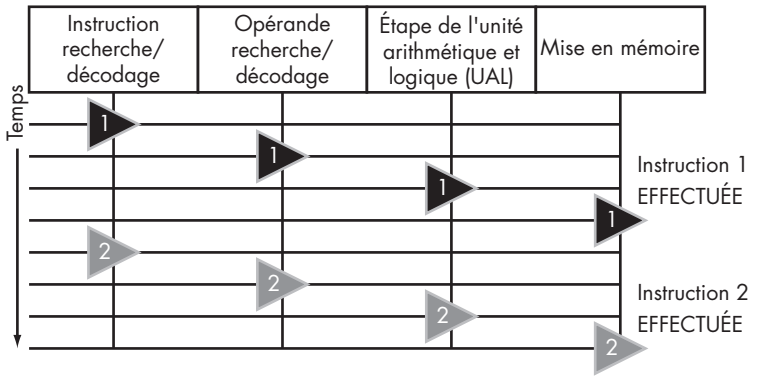


Figure 2.6

Étapes de base de l'exécution des instructions.

La Figure 2.6 illustre les étapes suivantes :

Instruction de recherche/décodage

Le processeur récupère une instruction de la mémoire, la traduit en une séquence de bas niveau, décide de la façon de traiter les données et des données à transmettre à toutes les étapes ultérieures. Le circuit est partagé pour toutes les opérations.

Opérande de recherche/décodage

Le processeur utilise un circuit générique pour collecter des opérandes à partir des sources de cette instruction en particulier (par exemple à partir des registres internes définis), de sorte que le circuit principal n'ait pas à prendre en charge toutes les combinaisons possibles d'opérandes et les stratégies de récupération.

UAL

Une unité arithmétique et logique (UAL) conçue spécialement pour effectuer cette opération en particulier, peut-être en plusieurs étapes, est appelée pour procéder à la tâche arithmétique définie. Pour les instructions (transfert de mémoire), des circuits génériques ou des circuits de l'UAL sont parfois utilisés pour calculer les adresses des sources et des destinations.

Mémoire

Le résultat est stocké à sa destination. Pour les opérations qui ne sont pas arithmétiques, la mémoire est copiée entre les emplacements calculés.

Seul, cela peut sembler n'être qu'une simple variante d'une exécution en plusieurs cycles classiques et la réutilisation d'un circuit de mesure, ce qui prévaut aujourd'hui pour la plupart des modèles de processeurs. Mais, comme vous le verrez, cela est également de la plus haute importance pour la vitesse d'exécution.

Le moins de mémoire possible

La simplicité des circuits n'est pas tout. La conception en plusieurs cycles présente également l'avantage de libérer la vitesse du processeur des limites de la mémoire, qui est l'élément le plus lent du système. La mémoire externe est considérablement plus lente que les processeurs actuels, a un temps d'accès et un temps de latence très élevés. Pour être fiable, un processeur monocycle ne peut pas être plus rapide que le temps nécessaire pour accéder à la mémoire, même si ce temps d'accès à la mémoire n'est pas continuel. Cette lenteur est nécessaire car une des instructions de son unique cycle *pourrait* exiger un accès à la mémoire, ce qui demande plus de temps. La conception en plusieurs cycles, en revanche, permet au processeur de ralentir, voire de se mettre en pause, pendant un certain nombre de cycles si nécessaire (au cours de l'entrée-sortie de la mémoire, par exemple), puis de passer à pleine vitesse lors de l'exécution de calculs internes. Dans une conception en plusieurs cycles, il est également plus facile d'accélérer les opérations qui sollicitent fortement la mémoire sans avoir à investir dans un accès plus rapide à la mémoire principale.

La conception flip-flop, communément appelée SRAM (mémoire vive à accès statique), offre un accès à faible temps de latence et consomme peu d'énergie. Les conceptions actuelles ont un temps de latence de 5 nanosecondes environ, ce qui est comparable à l'intervalle de temps entre deux cycles de certains processeurs. Malheureusement, cette conception exige aussi un nombre considérable de composants par flip-flop, environ six transistors par bits généralement.

Contrairement à la SRAM, la DRAM (mémoire vive dynamique à accès direct), l'autre type de mémoire couramment utilisé de nos jours, utilise une gamme de bascules pour mémoriser les informations. Ces bascules, toutefois, ont tendance à se décharger et doivent être mises à jour régulièrement. La DRAM nécessite plus de puissance que la SRAM et a un temps d'accès et un temps de latence pour modifier les données considérablement plus élevé, jusqu'à 50 nanosecondes. Mais la DRAM est beaucoup moins coûteuse à produire que la SRAM.

L'utilisation de la SRAM pour la mémoire principale est pratiquement inconnue en raison de son coût prohibitif. En outre, nous aurions des difficultés à utiliser toute cette augmentation des performances, car la mémoire devrait fonctionner à la même vitesse ou presque que le processeur. Hélas, comme la mémoire a une taille considérable et qu'elle est conçue pour être extensible, elle doit être placée à l'extérieur du processeur principal. Bien que le noyau du CPU puisse généralement fonctionner à une vitesse beaucoup plus élevée que le monde qui l'entoure, de graves problèmes de fiabilité (tels que la capacité des circuits de la carte mère, les interférences, le coût des puces périphériques à haute vitesse, et ainsi de suite) se posent lorsque les données doivent être transférées sur de longues distances.

Plutôt qu'utiliser de la mémoire externe plus rapide ou intégrer la totalité de la mémoire dans le processeur principal, les fabricants adoptent généralement une approche plus raisonnable. Les processeurs les plus évolués sont équipés de mémoires SRAM primaires rapides mais beaucoup plus petites ou de certains dérivés qui mettent en cache les régions les plus fréquemment consultées et stockent parfois certaines données spécifiques au CPU. Ainsi, chaque fois qu'un morceau de la mémoire se trouve dans le cache (*mémoire cache*), il peut être consulté rapidement. Ce n'est que lorsque le segment de mémoire doit être récupéré à partir de la mémoire principale (*défauts de cache*) que le délai peut être important et entraîner un retard considérable, à tel point que le processeur doit suspendre certaines opérations pendant un certain temps. (Les processeurs à cycle unique ne peuvent pas tirer pleinement profit de la mise en cache interne.)

Plus d'actions à la fois : le pipelining

Comme je l'ai déjà mentionné, le staging apporte en termes de performances un avantage considérable qui dépasse de loin l'approche multicycle traditionnelle. Il y a cependant une différence importante entre eux : comme la plupart des étapes sont partagées par diverses instructions, il n'y a aucune raison de ne pas optimiser l'exécution.

La Figure 2.6 montre que seule une partie de l'appareil est utilisée à chaque cycle lorsque les différentes étapes s'exécutent séparément. Même si l'instruction en cours a déjà réalisé les premières étapes de son exécution, elle bloque tout le CPU jusqu'à ce qu'elle soit terminée. Pour les systèmes dont le nombre de phases d'exécution est important (dix étapes ou plus pour les puces actuelles, les Pentium 4 dépassant les vingt étapes), cela représente un terrible gâchis de la puissance de calcul.

Une solution consiste à démarrer la première étape d'une instruction dès que l'instruction précédente passe à l'étape suivante, comme l'illustre la Figure 2.7.

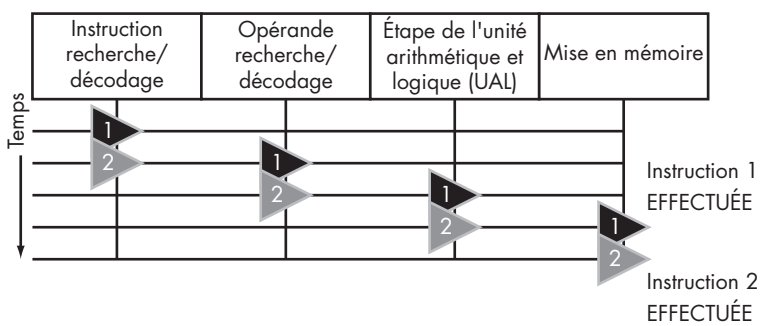


Figure 2.7

Modèle d'exécution du pipelining.

Dès qu'une étape particulière de la première instruction est terminée et que son exécution passe à laprochaine étape, l'étape précédente est alimentée par une partie de l'instruction suivante, et ainsi de suite. Au moment où la première instruction est terminée, l'instruction suivante n'est qu'à une étape de la fin de son exécution et la troisième, à deux étapes. Grâce à cette méthode en cascade, le temps d'exécution est donc grandement réduit et l'utilisation de la puce devient optimale.

Le pipelining fonctionne parfaitement tant que les instructions ne sont pas interdépendantes et n'utilisent pas le résultat de l'instruction précédente alors que celle-ci n'est pas terminée. Si les instructions dépendent les unes des autres, de graves problèmes s'ensuivent. En tant que tel, un circuit spécial doit être mis en place pour superviser le pipelining et empêcher ces conflits de dépendance.

Le pipelining présente d'autres difficultés. Sur certains processeurs, par exemple, le nombre d'étapes peut être différent suivant les opérations effectuées. Toutes les étapes ne sont pas toujours applicables, et il peut être plus optimisé d'en sauter certaines. Certaines opérations simples pourraient fort bien être exécutées beaucoup plus vite par l'intermédiaire du pipeline, car il n'y a pas d'opérandes à rechercher ou à mettre en mémoire. En outre, certaines étapes peuvent prendre un nombre variable de cycles, ce qui contribue aux risques de collision lorsque deux instructions atteignent le même stade de l'exécution en même temps. Pour éviter cela, certains mécanismes supplémentaires doivent être insérés, comme des instructions NOP (de l'anglais *No Operation*) conçues pour retarder temporairement l'exécution lorsque cela est nécessaire.

Le gros problème des pipelines

Le pipelining est un excellent moyen d'obtenir des performances élevées simplement en concevant des puces qui exécutent les instructions en plusieurs étapes, en réduisant le temps de latence des instructions ultérieures et en assurant l'utilisation optimale du circuit. Mais l'utilisation de pipelines n'est pas exempte de défauts : il n'est pas possible de l'utiliser pour des instructions après une instruction conditionnelle parallèle lorsque ces instructions risquent de modifier l'exécution des programmes.

En fait, cela est souvent possible, mais le processeur n'a alors aucune idée de la voie à suivre et, si une mauvaise décision est prise, tout le pipeline doit être vidé immédiatement après l'instruction parallèle (le CPU doit également retarder l'application de toutes les modifications apportées par ces instructions qui, après tout, ne devaient pas être exécutées). Les sauts conditionnels du pipeline provoquent un délai supplémentaire.

Et, malheureusement pour ce système, de nombreuses tâches qui sollicitent beaucoup le CPU, comme de nombreux algorithmes audio et vidéo, s'appuient sur de petites boucles conditionnelles exécutées des millions de fois à la suite, ce qui réduit considérablement les performances de l'architecture superscalaire.

La réponse à ce problème est la *prédiction de branchement*, réalisée généralement par des circuits de comptage assez simples qui suivent l'exécution du code le plus récent et utilisent un petit historique en mémoire pour estimer le résultat le plus probable d'une opération conditionnelle (bien que des conceptions plus complexes soient également souvent déployées[2]).

Cette exécution spéculative repose sur une stratégie conçue pour améliorer les performances du pipeline pour un code donné : si une instruction de branchement en particulier est plus souvent exécutée qu'elle n'est ignorée, il est alors préférable de rechercher et d'utiliser le pipeline pour les instructions. Bien sûr, l'estimation peut échouer. Dans ce cas, l'ensemble de la file d'attente doit être abandonné. Aujourd'hui, cependant, le taux de réussite de l'exécution spéculative est de 90 % pour du code classique.

Implications : de subtiles différences

L'ensemble des optimisations avancées employées dans les processeurs actuels a des conséquences intéressantes. Nous constatons que le temps d'exécution dépend des caractéristiques suivantes, qui peuvent être divisées en trois groupes :

- **Le type d'instruction et la complexité de l'opération.** Certaines opérations s'exécutent beaucoup plus rapidement que d'autres.
- **Les valeurs de l'opérande.** Certains algorithmes qui s'exécutent en plusieurs cycles se révèlent plus rapides pour les entrées simples. Par exemple, une multiplication par 0 est généralement simple et peut être effectuée rapidement.
- **L'emplacement de la mémoire à partir de laquelle les données nécessaires à l'instruction doivent être récupérées.** La mémoire cache est disponible plus tôt.

L'importance, la prévalence et l'impact de chacune de ces caractéristiques dépendent de la nature exacte de l'architecture du processeur en question. La première caractéristique, le temps d'exécution d'un nombre variable d'instructions, est partagée par toutes les architectures superscalaires mais peut être absente sur certaines puces basiques. La deuxième, qui dépend des opérandes, se retrouve de moins en moins sur les processeurs les plus évolués.

Dans les unités haut de gamme, l'UAL et l'unité de calcul en virgule flottante (FPU, de l'anglais *Floating Point Unit*) travaillent parfois à une vitesse supérieure à celle du processeur lui-même. Par conséquent, même s'il existe des différences de vitesse de calcul, elles ne peuvent pas être mesurées avec précision, car une grande partie du calcul est effectuée en un seul cycle d'horloge du processeur.

Le dernier groupe de motifs dans le temps, l'emplacement de mémoire, est à l'inverse l'apanage des ordinateurs actuels les plus performants et est totalement inconnu sur les contrôleurs bas de gamme et les diverses conceptions intégrées.

Les deux premiers groupes de motifs, la complexité des opérations et la valeur des opérandes peuvent également avoir une influence à un niveau légèrement plus élevé que le processeur, autrement dit au niveau du logiciel. Les fonctionnalités des unités de calcul des processeurs gèrent assez bien les petits entiers (généralement de 8 à 128 bits) et certains nombres en virgule flottante mais, aujourd'hui, la cryptographie et de nombreuses autres applications nécessitent la manipulation de nombres de grande taille (comprenant souvent des centaines de chiffres, voire des milliers), des virgules flottantes très précises ou diverses opérations mathématiques qui ne sont pas implémentées dans le matériel. Par conséquent, cette fonctionnalité est couramment implémentée dans les bibliothèques des logiciels. Les algorithmes de ces bibliothèques sont encore susceptibles de s'exécuter dans un temps variable, en fonction des spécificités de l'opération et des opérandes.

Utiliser les motifs dans le temps pour reconstruire les données

On peut envisager qu'un attaquant déduise certaines propriétés des opérandes ou d'une opération en surveillant le temps que prend un programme pour traiter les données. Cela constitue un risque de sécurité potentiel car, dans plusieurs cas, au moins un des opérandes peut être une valeur secrète qui n'est pas censée être divulguée à un tiers.

Bien que l'idée de récupérer des données en surveillant quelqu'un avec un chronomètre à la main puisse sembler surréaliste, les processeurs actuels disposent de compteurs précis qui permettent de déterminer précisément les intervalles de temps. De plus, certaines opérations peuvent être considérablement plus longues, certains codes opératoires avancés sur une plate-forme Intel prennent des milliers de cycles pour être menés à terme, par exemple. Le débit du réseau et les temps de réponse augmentant constamment, il n'est pas entièrement impossible de déduire ces informations, même à partir d'un système distant.

La nature des informations obtenues en mesurant la complexité des calculs peut ne pas être immédiatement évidente. Paul Kocher, de Cryptography Research, fit la preuve d'un excellent exemple de cette attaque au siècle dernier (c'est-à-dire dans le années 1990[3]), en utilisant un exemple de l'algorithme RSA dont nous avons parlé au Chapitre 1.

Bit par bit

Kocher a observé que le processus de décryptage des données dans l'algorithme RSA était assez simple, en se fondant sur la résolution de l'équation suivante :

$$T = c^k \bmod M$$

dans laquelle *T* correspond au message décrypté, *c*, au message chiffré, *k*, à la clé secrète et *M*, au modulo, qui est une partie de la paire de clés.

Un algorithme simple fondé sur la puissance d'un entier et utilisant l'opérateur modulo utilisé dans une application typique a une propriété importante : si un bit spécifique de l'exposant est un, une partie du résultat est calculée en effectuant la multiplication du modulo par une partie de la base (certains bits de *c*). Si le bit est 0, cette étape est sautée. Même si l'étape n'est pas vraiment sautée, le temps nécessaire au logiciel pour mener à bien des multiplications varie, comme indiqué plus haut. La plupart des cas simples, comme la multiplication à la puissance 2, sont résolus plus rapidement que d'autres.

Ainsi, sur un tel système, il semblerait qu'on puisse déterminer une multitude d'informations sur la clé (*k*) en vérifiant plusieurs fois le temps nécessaire au déchiffrement d'un élément de l'information. Même sur les plates-formes sur lesquelles la multiplication matérielle a une durée fixe, un motif dans le temps est souvent généré par les algorithmes de multiplication logiciels (comme l'algorithme de multiplication de Karatsuba) nécessaires au traitement de nombres de grande taille comme ceux utilisés par la cryptographie à clé publique. Les bits de l'exposant forment la clé privée, tandis que la base est une représentation du message fourni ou qu'une machine témoin peut voir.

Cette attaque est plutôt simple. L'attaquant envoie à la victime deux portions similaires mais légèrement différentes de données cryptées. Elles diffèrent dans une section X, de sorte que le déchiffrement de cette section prenne une durée différente. Si l'on considère ce que sait l'attaquant de l'implémentation de la multiplication par modulo par la victime, il lui est assez simple de trouver une des variantes de X et donc de déchiffrer rapidement X. L'autre variante est censée prendre plus de temps.

Si le temps nécessaire à l'attaquant pour décoder et réagir à ces deux séquences est identique, il peut supposer avec une quasi-certitude que la partie de la clé qui a été utilisée pour décoder la section X se compose de zéros. Il peut aussi supposer que l'algorithme de multiplication a pris le chemin d'optimisation le plus court, celui qui consiste à n'effectuer aucune multiplication.

Si, en revanche, l'un des scénarios prend plus de temps, il est évident que la multiplication a été réalisée dans les deux cas et que l'un d'entre eux est plus facile à résoudre. La partie correspondante du bit de la clé secrète doit avoir été fixée à une valeur non nulle.

À partir de cette procédure, en traitant les bits du message chiffré qui suivent comme notre "section X" et en générant, voire simplement en attendant (si l'on dispose de plus

de temps) des messages cryptés qui correspondent à ce scénario, il est possible de reconstruire tous les bits de la clé.

NOTE

Les recherches indiquent que cette approche peut être généralisée à tout algorithme qui s'exécute en un temps variable et examinent également comment optimiser l'attaque en pratique, en déployant une détection des erreurs limitée ainsi qu'une fonctionnalité de correction.

En pratique

La possibilité de déduire les propriétés tangibles des opérandes pour les instructions arithmétiques en se fondant uniquement sur le chronométrage des informations est le vecteur le plus évident, le plus efficace et le plus intéressant pour effectuer des attaques reposant sur des calculs complexes. D'autres techniques, comme celles fondées sur l'utilisation du cache et le temps écoulé, nécessitent généralement une analyse beaucoup plus détaillée et révèlent moins d'informations à chaque cycle.

Il est clair que ce problème touche, dans une certaine mesure, de nombreux algorithmes logiciels, comme les bibliothèques arithmétiques contenant des nombres de grande taille, qui sont couramment utilisées dans les applications cryptographiques. Mais, si l'on excepte les algorithmes logiciels et la théorie, deux questions importantes subsistent : quelle est vraiment la dépendance du temps d'exécution au niveau matériel, et comment la mesurer ?

Un exemple en est bien à portée de main. Au moins une partie de l'architecture Intel IA32 a ce comportement. Le *Manuel de référence du programmeur 80386*[4] décrit un code opération de la multiplication signée d'entier, désigné par le mnémonique IMUL. Le code opération, dans sa forme de base, multiplie la valeur stockée dans l'*accumulateur* (un registre multifonctions du nom [E] AX sur cette plate-forme) par une valeur stockée dans un autre registre. Le résultat est ensuite stocké de nouveau dans l'accumulateur.

La documentation donne en outre les explications suivantes :

> Le 80386 utilise un algorithme de multiplication early-out. Le nombre réel de cycles d'horloge dépend de la position du bit le plus significatif dans le multiplicateur d'optimisation (...). L'optimisation se produit pour les valeurs positives et négatives. En raison de l'algorithme early-out, le nombre de cycles d'horloge va du minimum au maximum. Pour calculer les vrais cycles d'horloge, utilisez la formule suivante :
>
> Actual clock = if m <> 0 then max(ceiling(log2(m)), 3) + 6 clocks
>
> Actual clock = if m = 0 then 9 clocks

Bien que cela puisse paraître obscur, la signification de cette formule est simple : le processeur optimise la multiplication en fonction de la valeur du multiplicateur. Plutôt que multiplier le multiplicande jusqu'à ce que tous les bits du multiplicateur soient épuisés, il saute les valeurs zéros au début de l'opérande.

Optimisation early-out

Pour comprendre la pertinence de cette tactique de multiplication des entiers, pensez à une méthode de multiplication itérative telle que celles qu'on enseigne dans les écoles, à la différence qu'il s'agit ici de binaire. Une possible implémentation "stupide" de cet algorithme effectue l'ensemble des opérations suivantes :

```
  00000000  00000000  11001010  11111110   Multiplicande (P)
* 00000000  00000000  00000000  00000110   Multiplicateur (R)
 ------------------------------------
  00000000  00000000  00000000  00000000   P * R[0] = P * 0
  00000000  00000001  10010101  1111110    P * R[1] = P * 1
  00000000  00000011  00101011  111110     P * R[2] = P * 1
  00000000  00000000  00000000  00000      P * R[3] = P * 0
  00000000  00000000  00000000  0000       P * R[4] = P * 0
  00000000  00000000  00000000  000        P * R[5] = P * 0
  ...
+ 0                                        P * R[31] = P * 0
 ----------------------------------------
  00000000  00000100  11000001  11110100
```

Il doit vous sembler évident qu'un grand nombre de ces opérations sont totalement inutiles et injustifiées et qu'il est tout simplement inutile de poursuivre l'opération lorsque les bits du multiplicateur sont tous égaux à zéro. Une approche plus raisonnable consiste à les sauter :

```
  00000000  00000000  11001010  11111110   Multiplicande (P)
* 00000000  00000000  00000000  00000110   Multiplicateur
                                             (R) - optimisation
 ----------------------------------------
  00000000  00000000  00000000  00000000   P * R[0] = P * 0
  00000000  00000001  10010101  1111110    P * R[1] = P * 1
+ 00000000  00000011  00101011  111110     P * R[2] = P * 1
  ...Bail out – ignore leading zeros of R!
 ----------------------------------------
  00000000  00000100  11000001  11110100
```

Voici, en substance, la nature de l'optimisation early-out qu'Intel a déployée.

NOTE

Cette optimisation rend la multiplication asymétrique dans le temps. 2*100 se calculera plus lentement que 100*2 (!), même si le résultat est évidemment le même.

En raison de cette optimisation early-out, les processeurs Intel ont besoin d'un nombre variable de cycles pour effectuer une multiplication, et sa durée est directement proportionnelle à l'emplacement du bit le plus ancien (le plus significatif) dans le deuxième opérande. En utilisant l'algorithme de comptage des cycles d'horloge fourni dans la documentation, il est possible de déterminer la corrélation entre le multiplicateur et le temps IMUL, comme illustré ici.

Plages de valeur du multiplicateur	***Cycles à compléter***
0 – 7	9
8 – 15	10
16 – 31	11
32 – 63	12
64 – 127	13
128 – 255	14
256 – 1023	15
1024 – 2047	16
2048 – 4095	17
4 096 – 8 191	18
8 192 – 16 383	19
16 384 – 32 767	20
32 768 – 65 535	21
65 536 – 131 071	22
131 072 – 262 143	23
262 144 – 524 287	24
524 288 – 1 048 575	25
1 048 576 – 2 097 151	26
2 097 152 – 4 194 303	27
4 194 304 – 8 388 607	28
8 388 608 – 16 777 215	29

(suite)

Plages de valeur du multiplicateur	*Cycles à compléter*
16 777 216 – 33 554 431	30
33 554 432 – 67 108 863	31
67 108 864 – 134 217 727	32
134 217 728 – 268 435 455	33
268 435 456 – 536 870 911	34
536 870 912 – 1 073 741 823	35
1 073 741 824 – 2 147 483 647	36

Une dépendance similaire existe pour les valeurs négatives du multiplicateur.

Code fonctionnel, faites-le vous-même

Le code suivant montre une implémentation concrète dans C pour les systèmes Unix-type qui peut être utilisée pour confirmer et mesurer les différences de motif dans le temps. Le programme est invoqué avec deux paramètres : *multiplicande* (qui ne devrait en aucune manière affecter les performances) et *multiplicateur* (on présume qu'il est utilisé dans les optimisations early-out et a donc un impact sur la vitesse de l'ensemble de l'opération). Le programme effectue 256 tests sur une suite de 500 multiplications avec les paramètres choisis et renvoie le plus court temps mesuré.

Nous exécutons 256 tests et choisissons le meilleur résultat afin de compenser les cas dans lesquels l'exécution est interrompue par le système durant une certaine période de temps, ce qui est assez courant dans les environnements multitâches. Même si un test peut être affecté par cet événement, on peut s'attendre à ce que certains des essais effectués dans une séquence rapide de tests s'effectuent sans interruption.

Le code utilise l'horloge du système pour mesurer le temps d'exécution en microsecondes.

NOTE

Plusieurs des puces Intel actuelles disposent d'un mécanisme de timing précis disponible par le code opération RDTSC. Cette méthode d'accès au compteur de cycles de l'horloge interne n'est pas disponible sur les anciennes plates-formes, si bien que nous ne pouvons pas compter sur lui.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <limits.h>

int main(int argc,char** argv) {

  int shortest = INT_MAX;
  int i,p,r;

  if (argc != 3) {
    printf("Usage: %s multiplicand multiplier\n",argv[0]);
    exit(1);
  }

  p=atoi(argv[1]);
  r=atoi(argv[2]);

  for (i=0;i<256;i++) {
    int ct;
    struct timeval s;
    struct timeval e;

    gettimeofday(&s,NULL);

    asm(

      " movl $500,%%ecx  \n"/*Boucle de répétition du compteur (R) */
      " imul_loop:       \n"
      " movl %%esi,%%eax \n"
      " movl %%edi,%%edx \n"
      " imul %%edx,%%eax \n"/* Commentaire à supprimer pour la première
        exécution */
      " loop imul_loop   \n"
    :
    : "S" (p), "D" (r)
    : "ax", "cx", "dx", "cc");

    gettimeofday(&e,NULL);

    ct = ( e.tv_usec - s.tv_usec ) +
         (e.tv_sec - s.tv_sec) * 1000000;

    if (ct < shortest) shortest = ct;

  }

  printf("T[%d,%d] = %d usec\n",p,r,shortest);
  return 0;
}
```

En compilant ce code avec l'instruction IMUL, qui n'est pas mise en commentaires la première fois, et en appelant le programme avec des paramètres arbitraires, on peut estimer la surcharge du code dans le temps (T_{idle}). Si cette valeur est inférieure à une fourchette de 10 à 100 microsecondes (ce qui est assez élevé pour fournir une lecture précise mais suffisamment bas pour minimiser les risques d'être interrompu par le système d'exploitation), ajustez la boucle de répétition de comptage R, qui est fixée à 500 par défaut.

Après le rétablissement de l'instruction IMUL, la recompilation et l'exécution du programme avec un multiplicande D et un compteur de répétition R définis, il est possible d'utiliser l'approximation de temps obtenue $T_{D, R}$ pour estimer le nombre de cycles processeur dépensés dans l'opération IMUL ($C_{D, R}$) tant que la fréquence de fonctionnement du processeur (F_{MHz}) est connue :

$$C_{D,R} = (T_{D,R} - T_{idle}). F_{MHz}/R$$

Comme on peut s'y attendre, les prévisions de branchement et de pipelining sur les puces les plus récentes et plus élaborées en fausseront légèrement le résultat, mais on peut tout de même obtenir une bonne estimation.

NOTE

Sur les processeurs Intel les plus récents, le temps nécessaire pour achever la multiplication est déjà constant.

Prévention

Plusieurs méthodes sont disponibles pour se protéger contre l'effort de l'analyse des efforts de calcul. La plus évidente consiste à faire en sorte que toutes les opérations s'exécutent dans le même laps de temps. Toutefois, cela est difficile et entraîne souvent une forte baisse des performances, car la durée de tous les calculs doit correspondre à celle du calcul le plus lent.

L'introduction de délais aléatoires est une tactique de défense acceptable si le temps de latence pour les applications n'est pas critique, comme dans le cas de plusieurs services réseau non interactifs, ce qui sollicite moins le processeur. Toutefois, ce bruit aléatoire peut être efficacement filtré si l'attaque est effectuée de façon répétée.

Une autre approche, connue sous le nom de *blinding*, repose sur l'introduction d'une certaine quantité de bruit dans le système en exécutant des bogues aléatoirement et des données imprévisibles combinées avec l'entrée réelle de l'algorithme afin que l'attaquant ne puisse pas déduire des propriétés significatives de l'entrée même si l'algorithme de chiffrement est vulnérable aux attaques dans le temps, puis en rejetant les données superflues que nous n'avons pas l'intention d'envoyer. Même si la baisse des performances est considérablement plus faible, il est difficile d'effectuer un *blinding*.

Matière à réflexion

Je vous ai entraîné dans un long voyage, mais j'espère que cela valait la peine. Comme d'habitude, je vous indique plusieurs problèmes assez intéressants à prendre en considération :

- Tout d'abord, même si je me suis concentré sur l'impact que les attaques sur la complexité des calculs ont sur la cryptographie, le problème ne se limite pas uniquement à ce domaine et se manifeste souvent dès que des informations privées ou confidentielles sont traitées. Bien sûr, diverses informations de base sur les requêtes HTTP ou le trafic SMTP peuvent être déduites en observant attentivement le service adéquat d'un système. Essayez d'envisager la mise en pratique d'autres scénarios.
- Deuxièmement, même si un service ne traite aucune donnée secrète, des informations sur la complexité des calculs peuvent être d'une certaine utilité. Prenez l'exemple des applications démon (ou *daemon*) sur le réseau, qui empêchent la divulgation de secrets en fournissant des messages (d'erreur ou de réussite) peut-être trop génériques. Le but de ces messages peut être de rendre difficile à un attaquant de savoir s'il obtient le message "connexion incorrecte" en raison d'une faute de frappe dans le mot de passe ou si l'utilisateur n'existe pas. Toutefois, selon le temps nécessaire à la réception de ce message, un observateur attentif peut déterminer quel chemin dans le code est effectivement exécuté, si l'erreur survient plus tôt (vérification uniquement de la validité du nom d'utilisateur) ou plus tard (vérification du mot de passe). Je vous encourage à expérimenter avec des services réseau comme SSH, POP3 et Telnet pour voir s'il existe une différence cohérente et mesurable.
- Comme toujours, même les meilleures défenses contre la divulgation d'informations ont tendance à échouer de façon inattendue. En outre, la complexité des calculs n'est pas la seule façon de déterminer ce qui se passe à l'intérieur d'une puce de silicium. Prenons l'exemple suivant : Shamir et Biham[5] ont élaboré un plan pour craquer les puces "sécurisées" utilisées dans les cartes à puce. Ces cartes à microprocesseur sont conçues pour stocker en toute sécurité des données d'identification personnelles ou des clés cryptographiques et de les divulguer uniquement à certains services d'authentification et à des clients de confiance. En fait, on peut déduire les propriétés des données ou le mécanisme de protection en trompant le dispositif grâce à des facteurs externes comme la fatigue mécanique, les rayonnements de haute énergie, une surchauffe et donc provoquer le mauvais fonctionnement de l'appareil.

Juste quelques pensées que je voulais partager.

3

Les dix têtes de l'hydre

Où l'on examine plusieurs autres scénarios qui surviennent très tôt dans le processus de communication.

Au cours des Chapitres 1 et 2, j'ai abordé deux scénarios distincts de divulgation d'informations qui se produisent à la suite de tentatives *a priori* brillantes mais finalement mal conçues de rendre les ordinateurs plus fonctionnels ou de faciliter leur maintenance. Les vecteurs d'espionnage passif que ces décisions de conception ouvrent sont en fait profondément enfouis sous l'implémentation et fournissent un aperçu fascinant sur les menaces qui pèsent sur les tout premiers traitements de l'information.

En revanche, cette vulnérabilité se limite naturellement à la proximité physique ou logique de l'environnement contrôlé. Même si les possibilités de divulgation de renseignements au début du traitement de l'information sont quasiment infinies, j'ai choisi de me pencher sur ces deux cas en raison de leur originalité, de leur beauté et de la facilité relative avec laquelle une attaque potentielle peut être effectuée par un attaquant déterminé. Cependant, les autres scénarios méritent également d'être signalés. Dans ce chapitre, j'aborde certaines des possibilités les plus intéressantes qui peuvent ne pas justifier une discussion exhaustive mais que vous pourriez avoir envie d'explorer plus en détail de votre côté.

TEMPEST : l'espionnage des émissions TV

Dans les années 1950, des études ont montré que le rayonnement électromagnétique peut en pratique souvent être utilisé pour récupérer ou reconstruire facilement des informations sur le comportement du dispositif qui l'émet. Le rayonnement électromagnétique est un bruit indésirable engendré par pratiquement tous les appareils électroniques, électromécaniques et électriques, indépendamment de leur conception et des objectifs qu'ils doivent remplir. Ce bruit est souvent propagé sur des distances considérables par les lignes électriques ou par voie aérienne.

Avant ces études, ce problème de rayonnement électromagnétique était considéré comme relevant du domaine de l'ingénierie en raison du risque d'interférences inattendues entre des dispositifs ou des circuits indépendants, mais sans intérêt pour une personne surveillant les fréquences radio émises par l'appareil. Toutefois, à l'aube de l'ère de la guerre de l'information, le développement croissant du traitement électronique des données et des appareils de télécommunication (certains utilisés pour transférer ou stocker des informations sensibles ou secret défense), les gouvernements des deux blocs s'inquiétèrent à l'idée qu'un observateur distant puisse reconstruire certaines des informations traitées par un système en écoutant simplement une fréquence spécifique.

L'acronyme TEMPEST (de l'anglais *Transient Electromagnetic Pulse Emanation Standard*) provient d'une étude secrète sur les rayonnements électromagnétiques menée par l'armée américaine dans les années 1960. Il désignait à l'origine un ensemble de pratiques visant à empêcher les émissions révélatrices d'informations dans les circuits électroniques qui traitaient des données sensibles. Ce terme devint plus tard un mot à la mode pour décrire l'ensemble des problèmes et des techniques ayant un rapport avec l'interception et la reconstruction des fréquences radio émises.

Même si, à l'origine, ce risque semblait davantage être de la mauvaise science-fiction que représenter une menace réelle pour les sceptiques, un important document de recherche publié en 1985 par Wim van Eck[1] a démontré qu'il était assez facile – et c'est en fait le cas – de reconstituer les images affichées sur un écran à tube cathodique (CRT) en interceptant les signaux des fréquences radio que les circuits à haute tension de ces moniteurs génèrent.

L'affichage sur un écran CRT classique (voir Figure 3.1) s'effectue en éclairant à très haute vitesse chaque pixel de l'image, ligne par ligne puis colonne par colonne, et en modulant constamment l'intensité du signal en fonction de la partie de l'écran qui est éclairée. Pour cela, une cathode à l'arrière de l'appareil émet un étroit faisceau d'électrons qui frappent l'anode (une couche de matériaux conducteurs sur l'écran), qui, à son tour, émet les photons de lumière visible que nous voyons. Le faisceau d'électrons est modulé par un circuit spécial mais aussi positionné par un ensemble d'électro-aimants pour

qu'il balaie l'ensemble de la zone d'affichage de gauche à droite et de haut en bas afin de produire et de rafraîchir l'image sur l'écran. Wim a noté que les oscillateurs contrôlant les électro-aimants et le canon à électrons émettaient plusieurs types de signaux caractéristiques à des fréquences standard. Il est simple de repérer ces signaux dans le spectre radioélectrique*. Chacun de ces signaux est généralement clair et suffisamment fort pour qu'il soit facile de construire un dispositif relativement peu coûteux capable d'espionner l'affichage des écrans CRT, même à une distance considérable.

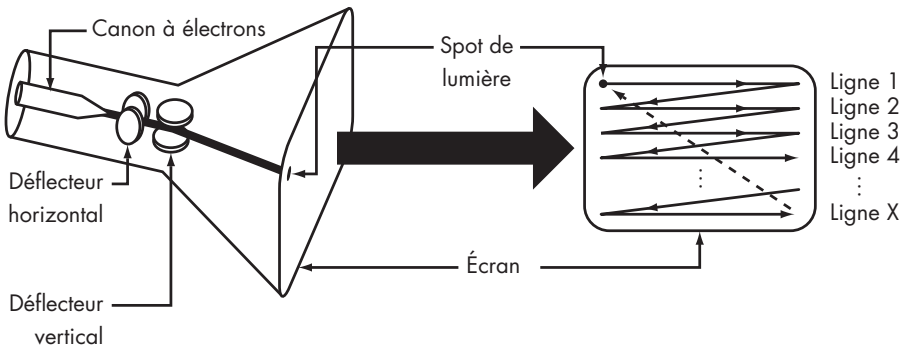


Figure 3.1

La création et l'affichage d'une image sur un écran cathodique.

NOTE

Bien entendu, les émissions ne se limitent pas aux écrans CRT et sont tout aussi courantes avec les écrans LCD (ou TFT, de l'anglais *Thin-film transistor*) et tous les circuits des ordinateurs. Elles sont tout aussi courantes sur les bus de données, là où l'information entre plusieurs puces distinctes quitte la carte mère en empruntant longuement des conducteurs qui, entre autres choses, agissent comme une grande antenne (bien qu'interpréter et extraire un signal spécifique ainsi que la plage d'une émission puissent être plus ou moins faciles).

* Pour cette raison, et à cause des interférences des lignes électriques, les personnes qui souhaitent écouter les signaux à très basse fréquence que produit naturellement la terre doivent souvent se déplacer avec leur matériel d'enregistrement dans des endroits isolés.

Même s'il n'existe pas d'exemples vérifiables d'attaques sur des émissions, à part pour des applications militaires et de renseignement (en particulier pendant la guerre froide[2]), certains documents rendent compte de son utilisation anecdotique pour l'espionnage industriel[3].

De toute évidence, ce genre d'attaque a ses limites: À moins d'espionner l'affichage d'un écran cathodique, l'attaquant doit se trouver à proximité de la cible et disposer d'un équipement complexe et très coûteux. Ceci est d'autant plus vrai s'il espionne les écrans à faible interférence ou les processeurs et les bus très rapides actuels. Pourtant, il est difficile et coûteux de se prémunir d'une telle attaque.

Les limitations de la confidentialité

Dans les scénarios examinés jusqu'ici, on peut considérer que les résultats indésirables ou inattendus ont pour cause la façon dont une technologie spécifique a été conçue et déployée, en dépit du fait que le développeur et l'utilisateur final partagent les mêmes objectifs. Dans certains cas, toutefois, l'exposition tient aux légères différences entre les objectifs et les attentes des deux groupes. Même s'il est fréquent que les problèmes de protection de la vie privée au niveau des logiciels soient dus à l'incompétence ou à la malveillance d'un programmeur, il existe également des problèmes de conception plus subtils qui ne sont pas un défaut en soi. Certains des problèmes les plus intéressants dans ce domaine entrent dans la catégorie de la divulgation des données dans les documents électroniques.

Nous supposons naturellement que, lors de la rédaction d'un document, toutes les informations ne se rapportant pas au contenu du document (et en particulier toute information qui identifie de manière unique son créateur) sont cachées aux autres personnes en mesure d'accéder à ce document, à moins que l'auteur ne les divulgue expressément. Mais les jours des éditeurs de texte non cryptés sont révolus depuis longtemps. De nos jours, les formats de document prennent en charge le stockage des métadonnées afin de marquer les documents pour ensuite les indexer, les rechercher et en opérer le suivi. En revanche, il est inquiétant que les concepteurs des outils de création décident souvent de remplir automatiquement certaines informations et ne donnent à l'auteur que peu ou pas de contrôle sur ce processus sans pour autant le tenir immédiatement au courant. Bien que cette pratique puisse être considérée comme un autre exemple de la volonté de rendre l'environnement plus convivial et transparent pour l'utilisateur, peu de personnes apprécient ce manque d'indications.

Établir la provenance des données

Un problème courant est que certains logiciels de création stockent des balises d'identification uniques qui permettent d'établir une corrélation entre un document et sa source. En particulier, Microsoft Word a longtemps utilisé l'adresse matérielle de la carte réseau (si l'ordinateur en possède une) pour construire un champ d'identifiant unique GUID (de l'anglais *Globally Unique Identifier*) dans un document – qu'il s'agisse d'une recette de cuisine ou d'un manuel de terrorisme. Bien que le problème ait été corrigé dans les versions les plus récentes de la suite d'applications Microsoft Office, cette pratique a eu des répercussions intéressantes :

- Chaque périphérique dispose d'une adresse de carte matérielle unique. Or les adresses matérielles sont utilisées pour localiser un périphérique spécifique sur un réseau local, ce qui est nécessaire pour éviter les problèmes qui se poseraient si deux ordinateurs ayant la même adresse matérielle se connectaient au même réseau. Le numéro enregistré dans le champ GUID d'un document Microsoft Word peut donc être utilisé pour identifier l'auteur du document, que cette personne ait écrit le document anonymement ou l'ait signé. Cela représente à la fois un précieux outil de recherche pour les enquêtes de police et un moyen efficace de réprimer la liberté d'expression dans certaines situations (par un employeur qui recherche les auteurs de certaines critiques, par exemple).
- Les adresses matérielles sont attribuées par lots à chaque fabricant. En outre, dans de nombreux cas, les numéros qui sont attribués aux cartes réseau le sont par ordre logique lors de leur fabrication, puis vendus par lots aux constructeurs. Ainsi, une personne peut non seulement connaître le fabricant d'une carte spécifique, mais également savoir qui l'a vendue et à qui. Dans de nombreuses situations, il est possible de suivre une adresse matérielle d'un ordinateur jusqu'à la personne ou la société qui le détient. Un enquêteur déterminé pourrait alors retrouver l'origine d'un document spécifique.
- Comme les adresses matérielles sont attribuées par lots, il serait également possible de tirer des conclusions limitées quant à la configuration matérielle du système sur lequel un document a été rédigé. Cela constitue une menace faible mais peut représenter une source intéressante d'informations pour les personnes qui s'amusent d'un rien ou qui sont particulièrement curieuses.

Certaines fonctionnalités, bien qu'accessibles à l'utilisateur, sont cachées si profondément au sein de l'interface qu'un utilisateur typique ne sait pas ce qui est enregistré ni la façon de modifier ces valeurs par défaut. Des logiciels comme Microsoft Word et OpenOffice sont connus pour insérer des informations sur "l'auteur par défaut". Ces informations correspondent généralement aux données fournies avec la licence du logiciel ou sont automatiquement enregistrées lors de la première exécution du programme à l'intérieur des métadonnées du document, là où la plupart des utilisateurs ne prennent

pas la peine de regarder. Même s'il s'agit d'un dispositif assez peu utile qui se révèle pratique lors du partage de documents, ses conséquences sur la confidentialité l'emportent généralement de loin sur les éventuels avantages que l'utilisateur final en retire.

On peut également prendre comme exemple la pratique "conviviale" qui consiste à remplir automatiquement le champ "titre" dans les métadonnées de l'en-tête d'un document à partir de la première phrase du document. Cela semble pratique mais ce choix est souvent permanent, ce qui signifie que, même si le premier paragraphe est changé par la suite (par exemple si l'entreprise fait une nouvelle offre à un concurrent de son premier client), le contenu original peut être déduit par un observateur attentif. Cette "fonctionnalité" dévoile encore une fois plus d'informations sur l'auteur que le destinataire ne devrait en savoir.

Les anciennes versions de Microsoft Word sauvegardaient également les documents sans effacer correctement toutes les données supprimées, ce qui fournissait donc des informations sur les annulations et enregistrait toutes les précédentes révisions du texte. Un attaquant suffisamment qualifié pouvait ensuite aisément récupérer ces informations avec un logiciel qui analyse les conteneurs OLE, le format dans lequel l'éditeur stocke toutes ses données. Ce problème est particulièrement grave lorsqu'une version antérieure d'un document est réutilisée comme modèle et envoyée à un destinataire différent qui peut être concurrent. La possibilité de récupérer la version précédente d'une offre, d'une lettre de motivation ou d'une réponse officielle à un client est certainement divertissante et instructive mais pas toujours souhaitable pour l'expéditeur.

Étant donné la tendance récente de l'informatique à réduire le piratage, il est raisonnable de s'attendre à ce que le balisage de tous les documents se banalise afin de pouvoir retrouver leurs auteurs.

Divulgation malencontreuse : *_~1q'@@... et le mot de passe est…

Le dernier type de problème que partagent bon nombre d'éditeurs de texte est celui de la fuite de mémoire aléatoire. Ce genre de divulgation est le résultat d'un manque flagrant de compétence ou d'un manque de tests, mais il se distingue des autres défauts du code dans la mesure où il ne rend pas tant le code vulnérable à une attaque qu'il ne divulgue des indications utiles pour un observateur attentif. Que ce problème se limite au programme seul ou soit causé par des fuites dans tout le système (comme sur les systèmes où la protection de la mémoire est mauvaise, comme Windows 3.x ou 9.x), cette fuite de données peut contenir des informations sensibles sur d'autres documents, sur l'historique de navigation, le contenu des e-mails ou même des mots de passe.

Le problème survient lorsque l'application alloue un segment de la mémoire (à un tampon d'édition, par exemple) qui a peut-être été utilisé auparavant pour une autre

tâche et qu'elle oublie de l'effacer avant de le réutiliser dans un tout autre but. Pour des raisons de performance, la mémoire n'est pas toujours remise à zéro avant d'être allouée à une application. L'application peut alors fonctionner sur une petite portion seulement du segment de mémoire et l'écraser, mais écrire sur l'ensemble du bloc de données alloué lors de l'enregistrement du fichier. Elle stocke alors à la fois les données prévues et du contenu résiduel présent depuis longtemps dans la mémoire et provenant d'on ne sait où. Et, cela n'a rien de surprenant, les anciennes versions de Microsoft Word étaient autrefois célèbres pour contenir aléatoirement des morceaux très importants de la mémoire dans quasiment tous les documents produits.

Ce problème a été observé un certain nombre de fois dans Microsoft Windows en 1998 sur tous les systèmes, puis sur Mac OS en 2001. Certaines données suggèrent que d'autres observations ont été effectuées, mais celles-ci sont assez mal documentées.

4

Travailler pour le bien de tous

Où la manière dont un ordinateur peut connaître les intentions de son utilisateur est soulevée sans obtenir de réponse.

L'avantage, mais aussi le gros problème de tout réseau informatique suffisamment vaste et diversifié est que vous ne pouvez pas aveuglément croire que la partie connectée soit réellement ce qu'elle prétend être et qu'il est impossible de connaître ses intentions réelles ou la motivation qui se cache derrière ses actes.

J'aborderai la question de la confirmation de l'identité d'une source dans la troisième partie de ce livre, quand je disséquerai l'architecture du réseau et étudierai les risques liés à la façon dont un réseau est conçu. Toutefois, la question des intentions de l'expéditeur est un aspect différent et fascinant de la sécurité informatique qui peut avoir des incidences sociales et judiciaires graves dépassant le monde de l'informatique. Comme les ordinateurs sont de plus en plus à même de prédire ce que l'utilisateur souhaite faire (pour rendre l'utilisation de l'ordinateur intuitive et plus facile) et qu'ils sont de plus en plus autonomes, il sont de plus en plus faciles à abuser et à utiliser par quelqu'un d'autre que l'utilisateur.

Quantité d'ouvrages ont été écrits sur ce sujet suivis par de nombreuses querelles enflammées pour définir à qui incombait la responsabilité et qui poursuivre en justice en cas de problème. Je crois qu'il est important de s'attaquer au problème, mais sans vouloir vous imposer un point de vue particulier. En tant que tel, je vais clore cette partie

de l'ouvrage avec la traduction d'un document court et principalement technique que j'ai publié initialement en 2001 dans le volume 57 du magazine *Phrack*. J'ai apporté quelques modifications mineures à cet article et je m'abstiendrai de plus amples commentaires.

Laissez-moi fouiller… chercher cet article… Ah, le voilà :

```
                              ==Phrack Inc.==
                  Volume 0x0b, Issue 0x39, Phile #0x0a of 0x12
 |=---------=[ Contre le système : Le soulèvement des robots ]=---------=|
 |=----------------------------------------------------------------------=[
 |=---=[ (C)Copyright 2001 par Michal Zalewski <lcamtuf@bos.bindview.com> ]=---=|

-- [1] Introduction -----------------------------------------------------

"... [La] grande différence entre le Web et les collections bien
contrôlées traditionnelles est qu'il n'existe quasiment aucun contrôle
sur ce que les gens peuvent mettre sur le Web. Si l'on ajoute à cette
possibilité de publier n'importe quoi l'énorme influence qu'ont les
moteurs de recherche pour acheminer le trafic, les entreprises qui
manipulent délibérément [sic] les moteurs de recherche dans un but
lucratif représentent un grave problème."

                                          -- Sergey Brin, Lawrence Page [A]

Imaginez un attaquant distant qui puisse compromettre un système sans
envoyer aucune ligne de code à la victime. Imaginez une attaque qui
créerait simplement un fichier en local afin de compromettre des
milliers d'ordinateurs sans impliquer aucune ressource spécifique.
Bienvenue dans le monde des techniques d'exploitation de bogue zéro
effort, de l'automatisation, de l'attaque anonyme, d'autant plus
difficile à éviter qu'Internet devient toujours plus complexe.

Ces exploits zéro effort créent une wishlist et la déposent quelque
part dans le cyberespace où d'autres peuvent la trouver. Ces "autres",
les travailleurs invisibles de l'Internet [B], sont les centaines de
robots infatigables qui ne dorment jamais et cherchent des
informations, les moteurs de recherche, les agents intelligents qui
viennent pour sélectionner cette information et deviennent à leur insu
un outil pour l'attaquant. Vous pouvez arrêter l'un d'eux mais ne
pouvez pas les arrêter tous. Vous pouvez découvrir ce que sont leurs
commandes, mais non deviner ce que ces commandes seront demain, car
elles sont cachées dans l'abîme toujours inexploré du cyberespace.

Ils forment une armée privée, toujours disponible et prête à obéir aux
commandes laissées sur leur chemin. On peut les exploiter sans avoir à
les compromettre. Ils font ce pour quoi ils ont été conçus, et ils le
font du mieux qu'ils le peuvent. Bienvenue dans une nouvelle réalité, où
les machines à l'intelligence artificielle peuvent se dresser contre nous.
Imaginez un ver. Un ver qui ne fait rien. Il est véhiculé et injecté
par d'autres, mais sans les infecter. Ce ver crée une liste de10 000
adresses aléatoires avec des commandes précises. Puis il attend. Les
```

agents intelligents indexent cette liste, unissant leurs forces pour toutes les attaquer. Imaginez qu'ils y réussissent avec, au pire, un taux de succès de 0,1 %. Dix nouveaux serveurs sont à présent infectés. Sur chacun d'entre eux, le ver prépare une nouvelle liste et les agents reviennent, infectant alors cent machines supplémentaires. Et l'infection se propage (en rampant, si vous préférez).

Les agents intelligents sont quasiment impossibles à percevoir, car les gens sont maintenant habitués à leur présence et à leur persévérance. Ils se contentent de progresser lentement, dans une boucle sans fin. Ils travaillent systématiquement, n'encombrent pas les connexions en générant des transferts de données excessifs, ne provoquent pas d'interruptions du réseau ou de pics d'activité et ne signalent pas la présence de la maladie. Semaine après semaine, ils inspectent prudemment de nouveaux hôtes, et leur exploration ne finit jamais. Et il possible de savoir qu'ils véhiculent un ver ? Peut-être...

-- [2] Un exemple --

Quand cette idée m'est venue à l'esprit, j'ai voulu vérifier si mon raisonnement était le bon à l'aide du test le plus simple possible. J'ai "ciblé", si tant est que le mot soit approprié, plusieurs moteurs de recherche et d'indexation généraux sur le Web. J'ai créé une page HTML très courte que j'ai déposée quelque part sur mon site et puis j'ai attendu quelques semaines. Et ils sont venus. Altavista, Lycos et des dizaines d'autres. Ils ont trouvé de nouveaux liens, les ont sélectionnés avec enthousiasme, puis ont disparu pendant plusieurs jours :

```
bigip1-snat.sv.av.com:
   GET /indexme.html HTTP/1.0

sjc-fe5-1.sjc.lycos.com:
   GET /indexme.html HTTP/1.0

[...]
```

Ils sont revenus plus tard, pour voir ce que je leur avais donné à analyser :

```
   http://somehost/cgi-bin/script.pl?p1=../../../../attack
   http://somehost/cgi-bin/script.pl?p1=;attack
   http://somehost/cgi-bin/script.pl?p1=|attack
   http://somehost/cgi-bin/script.pl?p1=`attack`
   http://somehost/cgi-bin/script.pl?p1=$(attack)
   http://somehost:54321/attack?`id`
   http://somehost/AAAAAAAAAAAAAAAAAAAAA...
```

Les robots ont suivi les liens, chacun d'entre eux simulant des vulnérabilités. Bien que ces exploits n'aient pas affecté mon serveur, ils pouvaient facilement compromettre des scripts en particulier ou un serveur Web distant en forçant le script à exécuter des commandes

arbitraires, à écrire des fichiers et pouvaient même entraîner un problème de dépassement de la mémoire tampon :

```
sjc-fe6-1.sjc.lycos.com:
   GET /cgi-bin/script.pl?p1=;attack HTTP/1.0

212.135.14.10:
   GET /cgi-bin/script.pl?p1=$(attack) HTTP/1.0

bigip1-snat.sv.av.com: GET /cgi-bin/script.pl?p1=../../../../attack
HTTP/1.0

   [...]
```

Les robots se sont ensuite joyeusement connectés aux ports non HTTP que j'avais préparés à leur intention et ont commencé à envoyer les données que j'avais fournies dans les URL, rendant ainsi possible une attaque sur d'autres services que ceux des serveurs Web :

```
   GET /attack?`id` HTTP/1.0
   Host: somehost
   Pragma: no-cache
   Accept: text/*
   User-Agent: Scooter/1.0
   From: scooter@pa.dec.com

   GET /attack?`id` HTTP/1.0
   User-agent: Lycos_Spider_(T-Rex)
   From: spider@lycos.com
   Accept: */*
   Connection: close
   Host: somehost:54321

   GET /attack?`id` HTTP/1.0
   Host: somehost:54321
   From: crawler@fast.no
   Accept: */*
   User-Agent: FAST-WebCrawler/2.2.6 (crawler@fast.no; [...])
   Connection: close

   [...]
```

En plus des moteurs de recherche bien connus, de nombreux robots de recherche et d'agents privés exécutés par diverses organisations et entreprises se sont également manifestés. Les robots de ecn.purdue.edu, de visual.com, de poly.edu, de inria.fr, de powerinter.net, de xyleme.com, et bien d'autres moteurs de recherche non identifiés ont trouvé cette page et l'ont appréciée. Bien que certains robots n'aient pas indexé toutes les adresses (certains agents n'indexent pas du tout les scripts CGI, d'autres n'utilisent pas les ports non standard), la majorité des moteurs les plus puissants ont attaqué virtuellement tous les vecteurs fournis. Même les plus prudents ont été abusés et ont effectué au moins quelques attaques.

Cette expérience pourrait être modifiée de façon à utiliser un ensemble de vulnérabilités réelles et entraîner des milliers et des milliers de débordements de serveur Web, des problèmes Unicode sur les serveurs Microsoft IIS ou des problèmes de script. Plutôt que cibler mon propre serveur, les robots pourraient pointer vers une liste d'adresses IP générées aléatoirement ou vers une sélection aléatoire de serveurs .com, .org ou .net. Ou, encore, on pourrait indiquer aux robots un service qui puisse être attaqué en fournissant une chaîne en entrée spécifique.
Il y a quelque part une véritable armée de robots de différents types, aux possibilités variables et plus ou moins intelligents. Et ces robots sont prêts à faire tout ce que vous leur demanderez.

-- [3] Considérations sociales ----------

Qui est coupable lorsqu'un agent "infecté" compromet votre système ? La réponse la plus évidente est : l'auteur de la page Web originale que le moteur a visitée. Mais il est difficile de trouver qui sont les auteurs des pages Web, et le cycle d'indexation d'un moteur de recherche prend des semaines. Il est difficile de déterminer quand la page en question a été mise sur le réseau car les pages peuvent avoir été déposées d'un grand nombre de façons, voire créées par d'autres robots. Il n'existe pas sur le Web de réel mécanisme de traçabilité qui fournisse des fonctionnalités semblables à celles implémentées dans le protocole SMTP. En outre, beaucoup d'agents ne se souviennent pas de l'emplacement où ils ont "appris" les nouvelles URL. L'utilisation d'indicateurs d'indexation, comme "noindex" sans l'option "nofollow", peut causer quelques problèmes supplémentaires. Dans de nombreux cas, l'identité de l'auteur et l'origine de l'attaque ne peuvent pas être totalement déterminées.

Par analogie avec d'autres cas, on peut s'attendre à ce que les développeurs intelligents de robots soient contraints de mettre en place des filtres spécifiques ou de verser d'énormes indemnisations aux victimes des abus des robots, si ce genre d'attaque devenait une réalité. En revanche, si l'on considère le nombre et la diversité des vulnérabilités connues, il semble presque impossible de réussir à filtrer correctement le contenu pour éliminer tout code malveillant. Et donc le problème persiste (de plus, tous les robots ne dépendent pas de la juridiction d'un seul et même pays et les lois sur les abus informatiques diffèrent sensiblement dans chaque pays).

-- [4] Défense ----------

Comme nous l'avons mentionné plus tôt, les agents et robots qui parcourent le Web disposent de moyens de défense et de possibilités d'actions très limités, en raison de la grande variété de vulnérabilités basées sur le Web. Il est tout simplement impossible de bannir toutes les séquences malveillantes et une enquête heuristique est risquée : une entrée valide et attendue par un script peut être suffisante pour attaquer un autre script. L'utilisation de logiciels sécurisés et correctement mis à jour est une des tactiques de défense les plus raisonnables pour toutes les victimes potentielles, mais cette

```
approche est très mal vue pour certaines raisons (voici un petit test
rapide sans valeur scientifique : la requête "cgi vulnerability"
retourne 62 100 enregistrements sur www.google.com avec le filtrage de
documents en place [C]). Une autre ligne de défense possible contre les
robots consiste à utiliser le mécanisme standard d'exclusion robots/
robots.txt [D]. Le prix à payer étant l'exclusion partielle ou complète
de votre site des moteurs de recherche, ce qui, dans la plupart des
cas, ne peut être envisagé. En outre, certains robots sont mal
développés ou conçus pour ignorer intentionnellement le fichier
robots.txt lorsqu'ils suivent un lien direct vers un nouveau site Web.

-- [5] Références -----------------------------------------------------

[A] "The Anatomy of a Large-Scale Hypertextual Web Search Engine",
    Googlebot concept, Sergey Brin, Lawrence Page, Stanford University.
    http://infolab.stanford.edu/~backrub/google.html


[B] "The Web Robots Database".
    http://www.robotstxt.org/wc/active.html


[C] "Web Security FAQ", Lincoln D. Stein.
    http://www.w3.org/Security/Faq/www-security-faq.html


[D] "A Standard for Robot Exclusion", Martijn Koster.
    http://info.webcrawler.com/mak/projects/robots/norobots.html


|=[ EOF ]=---------------------------------------------------------=|
```

Il semble pratiquement impossible d'empêcher totalement les abus automatisés si l'on n'est pas capable d'anticiper et de classer les intentions réelles qui se cachent derrière une action de l'utilisateur, ce qui n'est pas susceptible de se produire de sitôt. Pendant ce temps, le nombre de systèmes automatisés qui s'appuient sur l'interaction avec d'autres entités augmente chaque année, ce qui rend peut-être même cette question encore plus intéressante qu'à l'époque où j'ai écrit cet article, en particulier si l'on considère le nombre de vers de plus en plus sophistiqués qui ont frappé Internet au cours des dernières années.

Pouvons-nous tirer une morale ou une conclusion claire de cette histoire ? Pas vraiment. Il est cependant important de se rappeler que les machines n'agissent pas toujours au nom de leurs opérateurs, même quand ils ne sont pas clairement compromis ou réellement abusés et deviennent hostiles. Déterminer les intentions et l'origine d'une action malveillante intentionnelle peut constituer un formidable défi, comme vous le verrez dans les chapitres suivants.

Partie II

Un endroit sûr

Où il est question des menaces qui se cachent entre l'ordinateur et Internet.

5

Les Blinkenlights

Où l'on conclut que ce qui est joli peut également être mortel et où l'on apprend à déchiffrer les DEL.

La première partie de ce livre portait sur divers problèmes liés à la conception du système d'entrée des données. Ces problèmes se limitent à déduire l'entrée en observant les actions apparemment sans relation d'un utilisateur ayant un accès local à un système. Mais, lorsque les informations sont transférées au destinataire et quittent ce système, leur exposition augmente et les problèmes deviennent plus tangibles.

La deuxième partie de cet ouvrage met l'accent sur certains des problèmes qui surviennent alors que les données restent à portée de main juste après avoir quitté le système d'origine – le moment qui précède leur entrée sur Internet. L'exposition étudiée ici se limite à peu près à l'empreinte physique d'un réseau local et de son environnement immédiat. Une attaque à ce niveau exige un point d'observation local mais ne nécessite pas un accès au niveau du système.

Le problème particulier examiné dans le présent chapitre est quelque peu différent de ceux abordés précédemment : l'exposition se manifeste maintenant au niveau matériel, un peu comme dans TEMPEST, mais il est différent. La beauté de ce phénomène, ainsi que la facilité avec laquelle on peut l'observer sans disposer d'équipement spécialisé, justifie grandement qu'on l'examine de plus près.

L'art de transmettre des données

Dès les débuts de la pratique informatique, il était évident que les ordinateurs devaient communiquer avec d'autres appareils électroniques. Il était également manifeste qu'il serait difficile de rendre cette tâche fiable sans d'énormes investissements financiers. On peut contrôler les communications internes de la machine en plaçant entre tous les principaux composants différentes interfaces personnalisées à la capacité désirée qui conservent précisément les caractéristiques du signal et qui utilisent une horloge de référence commune pour toutes les opérations. Ainsi, le destinataire sait toujours quand être à l'écoute et l'expéditeur, quand transmettre les données. Mais le défi est tout autre pour les communications sur de longues distances ou vers des périphériques équipés d'interfaces non spécialisées et bon marché. L'ordinateur est alors obligé de communiquer sur un support qui n'offre généralement pas le degré de liberté auquel on est habitué lorsqu'on travaille sur les entrailles d'une seule machine.

En fait, la situation est même tout à fait inverse. Le client souhaite des solutions simples, pratiques et peu coûteuses, si bien qu'une connexion à l'aide d'un câble de 3 pouces composé de 100 fils et coûtant 100 dollars ne paraît pas être la bonne solution. La simplicité est nécessaire. À la base, tout canal de communication extérieure s'appuie presque toujours sur la transmission en série d'une suite de bits qui produisent uniquement des valeurs numériques, des chaînes de texte ou d'autres types de données natives lorsque ces bits sont réassemblés et regroupés sur la machine de l'expéditeur ou du destinataire. Dans l'exemple le plus simple, lorsque deux machines ou deux dispositifs connectés seulement par deux câbles doivent échanger des informations, ils le font en donnant à l'un des câbles une tension élevée ou basse en fonction de l'autre ligne (référence) ou en utilisant des signaux ou des états différents. Ceci afin d'envoyer des suites de bits de données à une fréquence donnée – une fréquence qui doit rester suffisamment proche et synchronisée sur les deux appareils.

Même pour une conception aussi simple, un certain nombre de problèmes se posent immédiatement. Premièrement, les périphériques ne partagent pas une horloge de référence. Bien que tous deux possèdent des horloges internes à base de quartz, deux horloges bon marché ne sont jamais assez précises pour garantir des communications rapides et fiables sur une longue période de temps, en raison de légers défauts de fabrication, des interférences et d'autres phénomènes physiques. Or une communication en série exige une synchronisation précise. Le système simple de codage des bits, généralement appelé NRZ (de l'anglais *Non Return to Zero*) produit uniquement en sortie un signal (une tension) pour 0 et un autre pour 1. Dans un tel système, il est facile de synchroniser les deux terminaux lorsque les valeurs changent régulièrement, le système n'ayant alors besoin que de détecter les deux états du signal, d'utiliser ces valeurs comme référence et d'ajuster sa propre horloge en conséquence. Mais, pour une longue séquence de 1 ou de 0, il devient difficile pour le récepteur de déterminer avec précision combien de bits sont envoyés.

En fait, même un petit décalage de l'horloge peut entraîner des problèmes, et il n'existe aucun moyen de les corriger lors de l'échange d'une séquence continue de bits.

La solution la plus évidente consiste à insérer dans les données un autre signal reconnaissable dans le temps. Mais cela n'est pas toujours la plus pratique ni la plus efficace. L'augmentation de la complexité et la réduction du débit sont souvent perçues comme une nuisance.

Pour répondre efficacement à ce problème, de nombreux systèmes utilisent un système appelé *codage Manchester* ou *codage biphase*. L'algorithme du codage Manchester, illustré avec le codage NRZ à la Figure 5.1, encode les données en utilisant les crêtes du signal, par opposition aux niveaux du signal. La version originale de l'encodage NRZ utilise une horloge interne pour mesurer les niveaux de tension à un rythme constant, en interprétant les tensions basses comme étant la valeur binaire 0 et les tensions hautes comme étant la valeur binaire 1. Le codage Manchester, en revanche, transfère la transition des données de l'état bas à l'état haut et *vice versa*. Le passage du signal à un état haut correspond alors à la valeur binaire 1 et sa transition vers un état bas, à la valeur 0*.

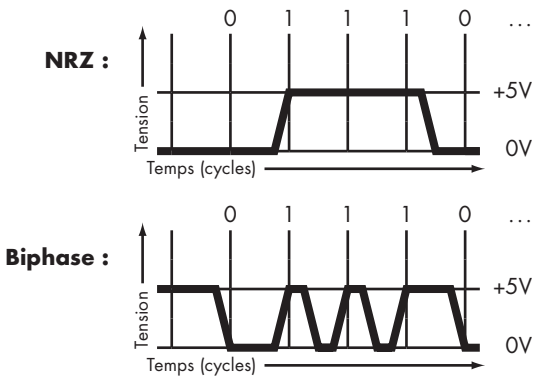


Figure 5.1

Codages des flux de transmission NRZ et biphase (Manchester).

Bien que ce codage n'exige pas que les horloges soient synchronisées, cela ne suffit pas : il n'y a aucun moyen d'encoder deux 0 ou deux 1 binaires, car il n'est pas possible de passer deux fois d'une tension faible à une tension haute (et *vice versa*) sans revenir à un état transitoire. Afin de permettre le codage de ce type d'information, les transitions qui se produisent peu de temps après une baisse ou une hausse du signal sont ignorées, ce qui permet au système de coder plusieurs occurrences de 0 et de 1 en revenant à la même tension en milieu de cycle. Pour gérer cette période de "trou" après une transition, un unique intervalle d'horloge est nécessaire.

---

* Ou l'inverse, selon la conception du transmetteur.

Ce schéma, fondé sur un schéma de synchronisation automatique, est souvent étendu pour fournir une liaison full-duplex dans laquelle les deux parties peuvent parler en même temps, en utilisant soit deux lignes séparées (transmission et réception, ou Tx et Rx) soit une détection avancée de l'écho et des astuces d'annulation pour différencier son propre signal des données envoyées par l'autre. Certains systèmes exigent ou permettent d'utiliser des signaux plus sophistiqués, comme l'envoi de plus d'un seul bit à chaque cycle, mais le principe des communications reste quasiment le même et le codage Manchester sur le nombre de fils le plus bas possible (souvent deux) prévaut.

À présent que nous connaissons les principes de base des communications en série par "paire torsadée", examinons deux bons exemples de communication en série dans le monde du travail en réseau, voyons comment s'échangent les données en interne et la manière dont des fuites d'information peuvent être divulguées à des tiers sans que l'utilisateur s'en aperçoive.

De votre courrier électronique à des bruits intenses… aller et retour

Le dispositif de communication longue distance le plus populaire est le modem. Créé au départ dans les années 1950 pour l'entretien et le contrôle de certains types de matériels militaires situés dans des endroits éloignés, le modem a rendu Internet accessible à tous. Même s'il est aujourd'hui souvent considéré comme obsolète, le modem a donné naissance à de nombreuses technologies de pointe, comme la technologie DSL (*Digital Subscriber Line*, ou ligne d'abonné numérique) ou le câble. Ces dispositifs utilisent tous des variations de la même technique pour communiquer sur les lignes téléphoniques analogiques ou sur d'autres médias non prévus à cet usage, à l'aide de signaux sonores ou inaudibles. Les recherches menées pour améliorer les modems ont également permis de mieux comprendre de nombreux problèmes de conception dans le domaine de l'électronique en général et dans celui de l'informatique et de la conception du réseau en particulier. Ainsi, il est primordial de comprendre le fonctionnement des modems pour explorer d'autres méthodes peut-être plus récentes de transmission de données sur de longues distances.

De par sa présence sur toute la planète, le réseau téléphonique représente le médium le plus logique pour établir des communications entre les ordinateurs. Comme les lignes téléphoniques se rencontrent presque partout et que les systèmes téléphoniques offrent d'excellentes capacités de routage des appels, il est ainsi possible de joindre à peu près n'importe quel endroit avec un minimum d'efforts. Un petit bémol, cependant : les lignes téléphoniques ont été conçues pour transporter la voix humaine, transmise sous la forme d'une onde, dans une fréquence de réponse étroite (ne dépassant pas généralement plusieurs kHz). Comme ces fréquences sont enregistrées en fonction des changements de tension sur une paire torsadée de câbles et relayées par un certain nombre de répétiteurs et d'amplificateurs analogiques, la qualité de la transmission n'est pas particulièrement élevée.

Il suffisait à l'origine que les gens s'entendent et se comprennent. Et, comme le cerveau humain est un magnifique système de filtrage et de traitement des signaux, ni le bruit occasionnel ni les fluctuations du niveau sonore n'étaient une préoccupation majeure jusqu'à ce que, bien plus tard, les clients deviennent un peu pointilleux.

Les ordinateurs, en revanche, sont généralement conçus pour échanger des informations binaires qui sont codées en utilisant des niveaux de tension assez précis sur des lignes courtes bien conçues et dont les caractéristiques de signal sont bonnes et la capacitance, faible. Autrement dit, l'exact opposé des lignes téléphoniques à longue distance, mal protégées et dont les caractéristiques du signal sont inadéquates. Les ordinateurs ont également besoin de parler beaucoup plus rapidement et beaucoup plus que les humains ne le font en général. Les concepteurs des modems avaient donc (et c'est un euphémisme) un défi difficile à résoudre : ils devaient trouver une façon non seulement d'encoder les bits de données afin qu'ils soient efficacement transmis par câble à un système distant (ce que le codage Manchester facilite un peu) mais aussi de le faire sous la forme de signaux audibles afin qu'ils soient reconnus avec précision à l'autre extrémité de la ligne sans tenir compte des changements de tension souvent totalement imprévisibles et des autres problèmes de transmission. Ils durent employer des algorithmes complexes de correction des erreurs et des vitesses de transmission variables pour compenser la mauvaise qualité de la ligne, le chevauchement occasionnel des conversations, le passage des camions sur une ligne téléphonique enterrée, la construction d'un nid d'oiseaux sur un poteau, et ainsi de suite. Il leur fallut environ quarante ans de recherche pour que nous disposions d'une technique de communication d'ordinateur à ordinateur abordable et assez rapide. Regardons rapidement le développement et la maturation de cette technologie (même si elle n'a pas radicalement changé) au fil des décennies suivantes.

L'histoire du développement commercial du modem et de sa normalisation a commencé dans les années 1960, lorsque deux normes, Bell 103/113 et V.21, ont été conçues. Ces deux normes fournissaient une connexion full-duplex à la vitesse étonnante (pour l'époque) de 300 bauds (bits par seconde) grâce à une technique appelée *modulation par déplacement de fréquence* (MDF), plus connue sous sa dénomination anglophone de *frequency shift keying* (FSK). Ce terme mystérieux désigne en fait un système simple de codage du signal : il utilise deux tons différents pour désigner des valeurs différentes, une fréquence pour la valeur "faible" et une autre fréquence pour la valeur "haute". L'utilisation de fréquences audibles présente un avantage assez important sur d'autres types de signaux : c'est en effet le seul type de signal qui peut être assez bien relayé par l'intermédiaire du système téléphonique puisque, après tout, c'est pour cela que le système a été conçu. Tous les autres signaux sont dans le meilleur des cas plus ou moins destinés à être supprimés car non reconnus avant d'avoir atteint l'autre extrémité du câble ou dans le pire des cas immédiatement éliminés par les filtres passe-bande quelque part dans la ligne.

En plus du codage MDF, les normes Bell 103/113 et V.21 divisaient la gamme de fréquences qui pouvaient être transmises par les lignes téléphoniques en deux : le

modem appelant utilisait une fréquence de 980 Hz pour encoder les valeurs basses et une fréquence de 1 180 Hz pour les valeurs hautes. À l'autre bout de la ligne, le répondeur utilisait la partie la plus élevée du spectre, soit 1 650 Hz et 1 850 Hz, pour chacune des valeurs. Pourquoi diviser la fréquence de cette manière ? Parce qu'une ligne téléphonique est avant tout une paire de fils qui peuvent uniquement être utilisés pour transmettre les données de deux appareils simultanément (full-duplex) s'ils sont capables de gérer le fait que chaque transmission se superposera à l'autre. Dans une communication en full-duplex, chaque dispositif doit être capable de distinguer son propre signal des données qu'il reçoit et les filtrer. Dans le cas contraire, chaque dispositif doit faire une pause lorsque l'autre parle (mode simplex), ce qui diminue fortement le débit déjà peu élevé. Par dédoublement de la fréquence, la ligne téléphonique transmet ce qu'elle considère comme deux "voix" et garantit donc que la communication se déroule simultanément sans collision.

Il a fallu vingt-cinq années supplémentaires pour que les modems franchissent une autre étape. La série suivante de normes importantes, Bell 212A et V.22, constituait un grand bond en avant et abandonnait la modulation par déplacement de fréquence au profit de la *modulation par sauts de phase* (DPSK, de l'anglais *differential phase shift keying*). Plutôt que modifier la fréquence d'une onde, DPSK modifiait sa phase pour signaler des valeurs différentes.

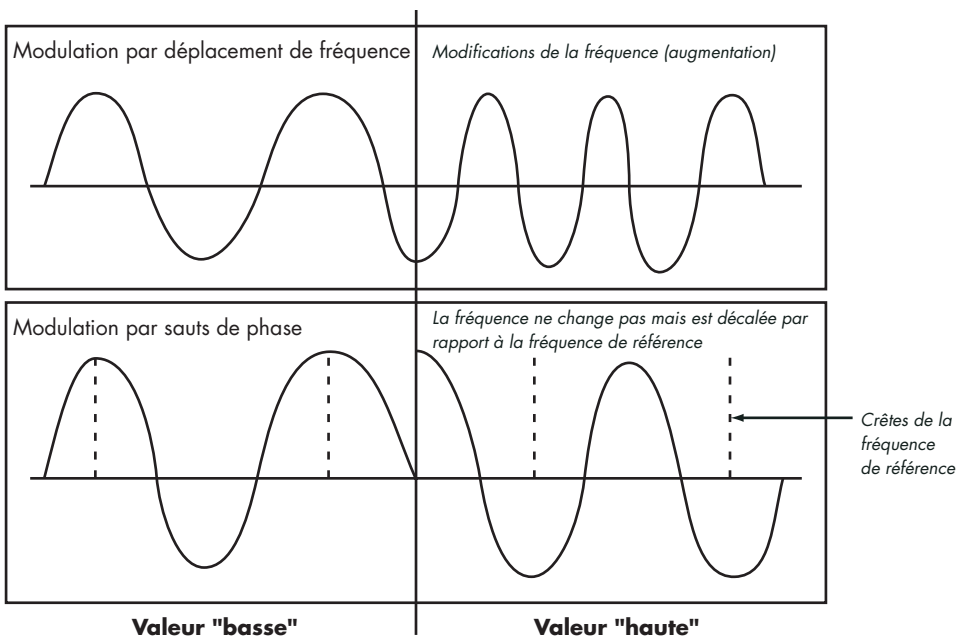

Modulation par déplacement de fréquence
Modifications de la fréquence (augmentation)
Modulation par sauts de phase
La fréquence ne change pas mais est décalée par rapport à la fréquence de référence
Crêtes de la fréquence de référence
Valeur "basse"
Valeur "haute"

Figure 5.2

Modulation par déplacement de fréquence et modulation par sauts de phase.

Cette technique de saut de phase entraîne un léger décalage horaire ou retard, si bien que le signal audio en sortie est légèrement désynchronisé avec l'onde de référence originale, tout en conservant exactement la même forme (voir Figure 5.2).

La valeur du saut de phase est exprimée en degrés, en référence à son effet sur les fonctions trigonométriques : y = sin(x) décalé de 90˚ est exactement identique à y = sin(90˚+x). Un changement de valeur de 360˚ dénote un saut de l'ensemble de la longueur d'onde, ce qui synchronise tout simplement de nouveau l'onde et n'a aucun effet sur la forme d'onde. La correspondance des différents sauts de phase est indiquée sur la gauche de la Figure 5.3.

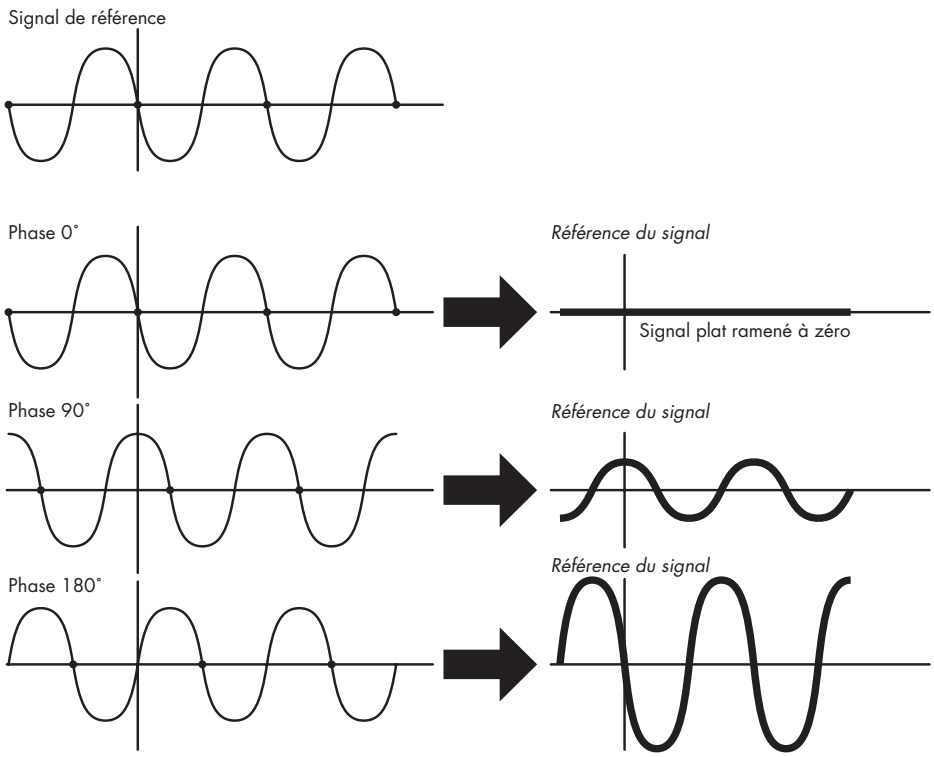


Figure 5.3

Les sauts de phase des signaux (à gauche) et le résultat après la soustraction d'une onde de référence afin de mieux distinguer les différentes phases (à droite).

Une fois que les deux parties sont synchronisées et disposent d'un moyen pour comparer le signal reçu sur le câble avec l'onde attendue, les données encodées peuvent être facilement récupérées. Un circuit différentiel peut comparer les deux signaux, les soustraire et aisément déterminer l'emplacement exact du saut de phase dans le signal en le comparant à un signal de référence, comme indiqué sur la droite de la Figure 5.3.

Cette nouvelle norme a également profité d'une méthode de codage des données plus avancée. Au lieu d'utiliser simplement deux signaux alternatifs pour transmettre des 0 et des 1, comme c'était le cas auparavant, la norme V.22 encode les bits par paire (familièrement appelées *dibits* en anglais). L'encodage de 2 bits à la fois peut être réalisé en utilisant quatre valeurs de saut de phase ; l'importance du saut est utilisée pour désigner chacune des valeurs possibles choisies afin que les valeurs soient uniformément espacées, si possible pour occuper les 360° du spectre, ce qui les rend plus faciles à distinguer les unes des autres (voir Tableau 5.1).

L'utilisation des dibits a permis d'accélérer sensiblement la vitesse de transfert (1 200 bauds), sans avoir à augmenter le taux physique auquel le signal est modulé. Deux fois plus d'informations, soit un nombre deux fois plus important de bits, sont transférées à l'intérieur de chaque signal.

Tableau 5.1 : Utilisation des sauts de phase pour encoder 2 bits de données (dibit)

Dibit	*Saut de phase*
00	90°
01	0°
10	180°
11	270°

NOTE

Bien qu'il soit en théorie également possible d'utiliser un alphabet aussi étendu, c'est-à-dire des unités de signaux composites semblables aux dibits (qui ont plus de deux états et donc encodent plus de 1 bit à la fois), avec le codage MDF, il est un peu plus problématique de le faire. Les signaux MDF doivent éviter les subharmoniques et les fréquences les plus susceptibles de subir une distorsion lors de leur transfert par l'intermédiaire des systèmes de téléphonie, ce qui limite énormément le nombre d'états possibles. La modulation par sauts de phase présente sur la modulation par déplacement de fréquence l'avantage d'utiliser une fréquence fixe connue pour causer moins de problèmes de transmission et qui peut par conséquent être utilisée de façon plus fiable pour des transmissions plus rapides.

Dans les années suivantes, le rythme des découvertes s'accéléra un peu, et un certain nombre de nouvelles normes firent leur apparition. La norme V.22bis utilisait de façon légèrement plus avancée le concept de l'alphabet de signaux, en combinant la modulation par sauts de phase avec une modulation de l'amplitude du signal (sa force) pour

construire un ensemble à deux dimensions de seize valeurs possibles. La transition d'un signal mesuré en valeurs binaires s'exprimait au moyen d'un tableau à deux dimensions. La valeur à laquelle correspond un signal s'obtient en recherchant d'abord dans la colonne, à partir de la mesure de la valeur du saut de phase, puis dans la ligne, en fonction de la mesure de l'amplitude. Le Tableau 5.2 illustre un exemple simplifié (4 colonnes pour 2 lignes) mais équivalent de ce tableau.

Tableau 5.2 : L'encodage en deux dimensions de 3 bits utilisant deux paramètres de signal distincts

	Phase 0˚	***Phase 90˚***	***Phase 180˚***	***Phase 270˚***
Amplitude faible	000 (0)	001 (1)	010 (2)	011 (3)
Amplitude élevée	100 (4)	101 (5)	110 (6)	111 (7)

Pour rendre les choses un peu plus confuses, cette nouvelle approche fut baptisée *modulation d'amplitude en quadrature* (ou QAM, de l'anglais *quadrature amplitude modulation*). Elle permit de passer de 1 200 à 2 400 bps sans réellement améliorer la vitesse de modulation du signal, mais en élargissant le nombre de significations qu'un seul signal pouvait avoir.

L'évolution suivante la plus importante fut la norme V.32. Elle fut la première à introduire un nouveau concept : au lieu de diviser les fréquences, elle utilisait des circuits d'annulation de l'écho* pour détecter et soustraire le signal transmis par le périphérique lui-même à partir des données reçues du câble. Cette technique permettait aux deux appareils (émetteur et récepteur) d'utiliser l'ensemble du spectre des fréquences, au lieu de la moitié seulement, tout en conservant toujours le full-duplex.

Le développement se poursuivit rapidement avec l'apparition du protocole V.34. Même si la vitesse à laquelle le signal pouvait être alterné en toute sécurité avant que des distorsions excessives n'apparaissent ne fut pas sensiblement modifiée au cours des années, cette norme était considérablement plus rapide que ses prédécesseurs. Les modems V.34 atteignaient un débit de 28 800 bauds, parfois même la vitesse non officielle de 33 600 bauds (33,6 Kbit/s) chez certains fabricants, en envoyant seulement environ 2 500 à 3 500 échantillons de signaux (symboles de l'alphabet) par seconde. Toutefois, cette norme combinait

* Les circuits d'annulation de l'écho tentent de distinguer les signaux en provenance de l'autre partie de ceux émis par le périphérique lui-même et d'éliminer ou de réduire de façon significative ces derniers. Différents types de ces dispositifs sont couramment utilisés non seulement lors du transfert de données numériques, mais aussi pour améliorer la qualité des appels téléphoniques, éliminer les interférences des microphones au cours des manifestations publiques et résoudre de nombreux autres problèmes quotidiens.

quatre systèmes de codage différents pour construire une structure à quatre dimensions avec 1 664 états possibles, ce qui rendait possible d'envoyer jusqu'à 41 bits à la fois. En fait, il s'agit non pas de vitesse pure mais de la façon d'utiliser ce dont vous disposez.

Il est largement admis que la norme V.34 et ses dérivés approchent de la limite théorique de transmission des données par le système téléphonique prévu pour la voix. Bien que cela puisse sembler étrange étant donné la prédominance des modems 56 Kbit/s, cela s'explique du fait que les modems 56K atteignaient ce taux de transmission d'une manière totalement différente de celle utilisée dans les solutions analogiques. Étant donné que la plupart des systèmes téléphoniques ont migré de l'analogique au numérique depuis l'apparition des premiers modems et que la plupart des fournisseurs d'accès peuvent désormais insérer directement une interface entre leurs systèmes et les systèmes de télécommunication numérique, il devient enfin possible de revenir à la solution la plus évidente : modifier la tension des lignes au lieu de moduler les fréquences lors de l'envoi des données à un abonné. Le signal est transporté sous forme de données numériques depuis le début et peut voyager sur des lignes de cuivre enterrées jusqu'à la plus proche installation de télécommunications, la qualité du signal ne pose pratiquement pas de problème et la seule limite tient à la capacité du matériel téléphonique à transférer la voix. En travaillant sur 8 000 symboles par seconde mais en exploitant un alphabet beaucoup plus réduit (environ 128 symboles ou niveaux de tension généralement), il est possible d'envoyer des données à un abonné relié à un système téléphonique numérique de haute qualité avec un modem de 56 Kbit/s à une vitesse plus élevée que d'habitude. Cependant, le transfert en amont est encore implémenté avec l'ancienne méthode, et il est considérablement plus lent. Par conséquent, le débit du modem est en partie seulement de 56 Kbit/s, et uniquement lorsque les conditions le permettent.

De nos jours

Peu de choses ont changé depuis la conception des modems. Les avancées des protocoles s'accompagnaient également de l'amélioration des mécanismes de correction des erreurs et de réduction de la vitesse de transfert, qui étaient nécessaires pour assurer une transmission fiable, même si votre quadrupède favori décide de mâcher le câble du téléphone. Une jungle des normes a été engendrée : V.42 offrait une implémentation basique du contrôle de redondance cyclique (CRC), MNP-1 à MNP-4, des algorithmes propriétaires de correction des erreurs, V.42bis et MNP-5, le contrôle de l'intégrité combiné à la compression, etc. Mais la véritable révolution est encore à venir.

Ou ne s'est-elle pas déjà produite ? Certains considèrent en effet que les modems DSL et le câble forment une technologie révolutionnaire qui a changé la face du monde. Mais je suis prêt à défendre mon point de vue : en fait, ces technologies sont très semblables à leurs cousins les modems. La seule différence importante entre les deux est que le serveur qui gère toutes les connexions a été déplacé de la ville lointaine où le prestataire de services se trouve au central téléphonique local le plus proche auquel il est possible de se connecter directement depuis la résidence du client ou de l'entreprise en utilisant des fils de cuivre. Comme cette connexion directe n'utilise aucun autre équipement, ces dispositifs peuvent utiliser des fréquences élevées et inaudibles ainsi que des signaux plus fins qui, autrement, subiraient une distorsion ou ne seraient pas relayés sur tout le réseau téléphonique. À l'inverse, le bon vieux modem était strictement limité à l'étroite gamme de fréquences audibles et de signaux que le système téléphonique pouvait transporter. À de nombreux égards, les dispositifs DSL ont une tâche beaucoup plus facile que les anciens modems.

Comme nous le voyons, la conception d'un modem est plutôt une tâche complexe et difficile. C'est pourquoi il a fallu des décennies pour passer des périphériques à 300 bauds volumineux et coûteux à la situation actuelle[1]. Curieusement, tous ces dispositifs peuvent communiquer les uns avec les autres, même avec ceux qui ont dix ans de plus, même aux vitesses les plus basses que nous avons oubliées depuis longtemps. En outre, tous sont généralement conscients des normes connues à ce jour, y compris les dizaines d'alternatives et de dérivés de chacune d'entre elles. Cela ne fait-il pas des modems une merveille de l'ingénierie informatique ?

Mais qui tire les ficelles ?

Parfois, un modem est juste un modem

L'histoire ne se résume bien sûr pas aux communications de modem à modem. Le modem est juste un intergiciel relativement inerte à peine capable d'être un bon presse-papiers. Pour qu'un modem soit d'une quelconque utilité, il doit être capable de communiquer avec un ordinateur afin de recevoir des commandes et d'échanger des données, même si son utilisation se résume à quelque chose d'aussi futile que la navigation au hasard sur le Web. Les modems internes ont la vie facile : ISA (Integrated Systems Architecture), PCI (Peripheral Component Interconnect), PCMCIA (PC Memory Card International Association) et quelques autres bus dédiés fournissent des interfaces parallèles à grande vitesse et assez généreuses qui rendent le processus de communication presque banal.

Les modems externes (de type analogique ou DSL), en revanche, ont une tâche plus ardue et utilisent une liaison série. La plupart des modems analogiques utilisent le célèbre protocole série RS-232 (renommé EIA/TIA-232-E2[2] dans les années 1990, ce qui

est beaucoup plus descriptif), les plus récents utilisent fréquemment le port USB (Universal Serial Bus). Comme nous approchons des exemples de divulgation d'informations pour ces dispositifs, nous allons également examiner ce qui arrive aux données lors de leur passage du modem à l'ordinateur, car cela joue un rôle crucial dans l'attaque.

Bien que les modems externes doivent utiliser des moyens inhumains pour communiquer avec un système distant mais aussi avec la machine locale elle-même, grâce à la proximité de l'ordinateur et au fait que des interfaces comme RS-232 sont numériques et ont été conçues dès le départ pour être utilisées par les ordinateurs, cette étape est toujours beaucoup plus simple que la modulation et la démodulation de la ligne téléphonique qui rendit célèbres les modems.

RS-232 utilise une implémentation assez simple de codage bipolaire pour les données échangées sur deux lignes séparées couplé avec un ensemble de lignes de contrôle NRZ. Pour rendre la vie un peu plus intéressante, RS-232 est livré avec une multitude de fonctionnalités de liaison ou de protocole qui le rendent assez difficile à mettre en œuvre à partir de zéro : son caractère asynchrone, le large éventail de paramètres et de vitesses possibles, et des niveaux de tension inhabituels. Mais, même avec tout cela, RS-232 ne constitue même pas l'ombre d'un véritable défi pour un implémenteur qui a dû batailler avec la modulation du signal sur les lignes téléphoniques.

L'USB, par ailleurs, cherche à normaliser et à unifier l'interface série. Bien que l'USB requière des circuits plus évolués que RS-232 pour interfacer un ordinateur et un périphérique (en raison entre autres choses du plus haut niveau d'abstraction et des vitesses de transmission plus élevées prises en charge), l'USB est universel (d'où son nom) et a hérité de moins de fonctionnalités bizarres.

Dernier point mais non le moins important, l'utilisation d'Ethernet représente une méthode commune pour communiquer avec des périphériques locaux. Bien qu'il soit antérieur à l'USB, son mécanisme est assez similaire. Penchons-nous maintenant et pendant un certain temps sur Ethernet, et je suis sûr que tous ces protocoles de communication finiront par se rejoindre.

Les collisions sous contrôle

Les réseaux Ethernet sont, par nature, une forme avancée de liaison série multipartite[3]. Un réseau Ethernet est composé d'un certain nombre d'ordinateurs connectés par le même moyen – rien de particulièrement complexe puisqu'il peut s'agir uniquement de câbles à paire torsadée. Quand un périphérique sur le réseau utilise ce médium, il applique une tension spécifique sur le câble. Tous les autres systèmes connectés peuvent alors interpréter les données en mesurant la tension. Une série de contrôle

veille à ce que les périphériques ne cherchent pas à utiliser cette liaison en même temps et que la restauration de la connexion se fasse sans heurt en cas d'accident. Néanmoins, même en considérant cette possibilité, la conception de base est incroyablement simple comparé aux modems.

Pour éviter que deux stations ne parlent à la fois, une norme baptisée CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) est utilisée en tant que principal mécanisme de contrôle de toutes les communications *via* Ethernet. Avant tout envoi de données, chaque périphérique connecté à Ethernet suit une procédure CSMA pour voir si un autre périphérique utilise le câble en vérifiant les propriétés électriques du modem. Si aucune autre transmission n'est en cours, le périphérique entre dans la phase de la transmission et envoie ses données sur le réseau.

Les données sont alors envoyées sur le câble sous la forme d'une séquence de bits en utilisant le *codage bipolaire* ; un en-tête contient toutes les informations nécessaires sur l'expéditeur et le destinataire ainsi qu'une somme de contrôle destinée à protéger l'intégrité des données en cas d'interférence extérieure ou intérieure. Une interface réseau qui considère qu'elle agit au nom d'un destinataire, probablement en comparant l'adresse de destination fournie dans le paquet avec l'adresse MAC (matérielle) unique stockée sur la carte, devrait accepter ce transfert et vérifier la somme de contrôle. En même temps, toutes les autres stations devraient l'ignorer. Naturellement, si elles ne le font pas (presque toutes les cartes peuvent avoir pour instruction de ne pas le faire), l'utilisateur peut afficher ou réagir aux transferts qui ne lui sont pas destinés (vous pouvez constater que la conception des réseaux Ethernet repose sur l'altruisme et la confiance, une démarche noble mais risquée).

Il est possible (et pas si improbable) que deux périphériques reliés à un réseau Ethernet commencent à émettre exactement au même moment, même si les deux ont vérifié quelques microsecondes ou nanosecondes avant qu'aucune transmission n'était en cours. Et, s'ils émettent exactement au même moment, une catastrophe est inévitable. Les deux transmissions se mélangent et sont altérées, si bien que les données envoyées ne passent pas la somme de contrôle lors de leur arrivée à destination… Le devraient-elles ?

Bien que l'implémentation d'une somme de contrôle dans la spécification des trames Ethernet soit généralement suffisante pour vérifier l'exactitude des transmissions de données, cela peut ne pas être particulièrement efficace si la liaison est saturée et que des centaines ou des milliers de collisions se produisent dans un laps de temps suffisamment court pour que la sortie soit accidentellement correcte de temps à autre. Selon la loi de probabilités, certains paquets endommagés auront par hasard la même somme de contrôle que le paquet initial. En outre, même si l'on ignore les lacunes de la somme de contrôle, on souhaite quand même faire cesser les collisions le plus rapidement

possible car, en laissant les collisions s'exécuter en arrière-plan, on risque de ne plus être en mesure d'assurer dans les délais la retransmission des trames erronées et abandonnées. Après tout, l'expéditeur les envoie sans indiquer de problème et le destinataire ne reçoit rien qui ressemble même de loin à un paquet utile.

La solution est fournie par la dernière partie de la norme : la détection de collision (CD). Cette spécification demande à l'expéditeur de surveiller la liaison réseau tout en expliquant ce qu'il fait aux autres. Si une autre partie est prise en train d'essayer de parler en même temps, cela devrait être détecté (encore une fois par une simple mesure des propriétés électriques de la ligne) et la transmission, être immédiatement abandonnée. Le périphérique devrait également envoyer un brouillage du signal spécial pour garantir que les deux trames (celle qui est envoyée et celle qui interfère) soient abandonnées sans condition, sans même arriver à être vérifiées par la somme de contrôle ; le destinataire doit être en mesure de repérer le brouillage du signal et d'arrêter la réception des données en cours de traitement. Le périphérique se met alors en pause pendant une période de préférence aléatoire (initialement) qui augmente progressivement après chaque tentative (définie par un algorithme de backoff) afin de réduire au minimum la probabilité d'une autre collision.

NOTE

Un fait amusant : le mécanisme du brouillage du signal impose une exigence inhabituelle sur le protocole. Toutes les trames doivent avoir une longueur minimale (!) calculée pour que le brouillage du signal généré soit propagé à toutes les machines avant la fin de la transmission. Avec des trames très courtes, le temps peut ne pas être suffisant pour y parvenir. Ainsi, il est demandé à l'expéditeur de bourrer artificiellement l'ensemble de ses transmissions sortantes.

La Figure 5.4 montre la chronologie exacte des événements dans un cas typique de collision. Comme vous pouvez le voir, l'expéditeur A espère envoyer des données au destinataire mais remarque qu'une autre transmission a lieu et décide donc d'attendre jusqu'à ce que cette transmission s'arrête. L'expéditeur A se prépare ensuite à envoyer les données mais, malheureusement, l'expéditeur B fait de même, si bien que les deux pensent à peu près en même temps qu'ils peuvent envoyer des données.

Les deux tentent de transmettre, les données sont altérées, les deux expéditeurs détectent l'autre transmission et envoient rapidement un brouillage du signal pour signaler au destinataire qu'il ne doit pas tenir compte de cette trame. Enfin, les deux expéditeurs se mettent en pause pendant une durée aléatoire en espérant ne pas commencer à émettre en même temps la prochaine fois.

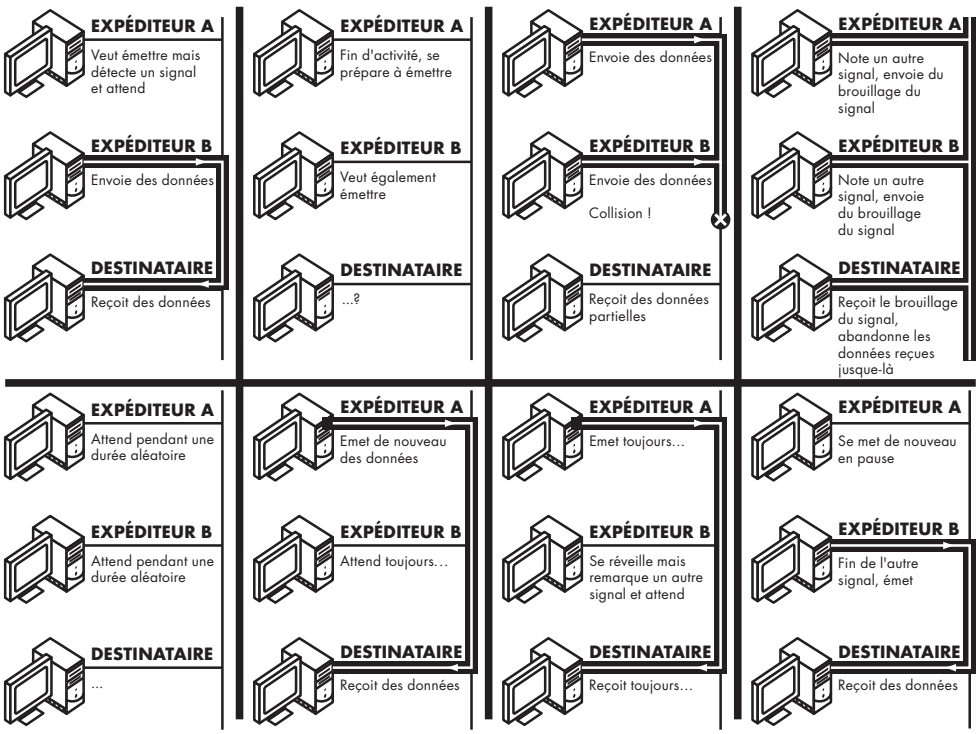


Figure 5.4

Les phases classiques d'une conversation Ethernet.

Les coulisses : la soupe de câble et comment la gérer

Bien que sa conception ne soit ni particulièrement évolutive ni élégante, le protocole Ethernet est étonnamment puissant et facile à déployer ; il permet de créer des réseaux de communication de station à station bon marché en utilisant des câbles coaxiaux à peu près partout. À ce titre, Ethernet est devenu la norme en remplacement de beaucoup d'autres architectures de réseaux (qui lui sont parfois supérieures mais plus coûteuses ou propriétaires).

Naturellement, un réseau Ethernet simple qui utilise des câbles coaxiaux a des limites et des inconvénients. Il se compose essentiellement d'un long câble comprenant des résistances à chacune de ses extrémités et le long duquel se branchent les périphériques. Autrement dit, un système dont vous ne voudriez pas assurer la maintenance dans un grand bureau. Un problème classique mais difficile à déboguer, comme le court-circuit d'un terminal, peut mettre en panne l'ensemble de l'infrastructure. Son remplaçant plus avancé mais seulement légèrement plus cher fut donc chaleureusement accueilli.

Les répéteurs électroniques multiports (concentrateurs, ou hubs) permettent d'utiliser le câblage sur paire torsadée (câbles de catégorie 3 et catégorie 5 avec connecteurs RJ-45). Pour les utiliser, il suffit de relier votre ordinateur à une boîte noire avec un câble. Tous les autres appareils reliés à cette boîte noire peuvent alors communiquer avec lui sans vraiment tenir compte d'éventuels problèmes électriques ou courir le risque qu'un problème sur un des câbles ne mette en danger l'ensemble du réseau.

Les concentrateurs sont pour l'essentiel de simples répéteurs qui diffusent tout le trafic reçu sur un port à tous les autres ports. Ils permettent de créer des réseaux en étoile facilement reconfigurables et plus fiables, mais rien de plus. À mesure que le réseau se développe, le coût de la diffusion de chaque élément d'information à tous et le fait que seule une partie peut parler à la fois sur l'ensemble du réseau montrent à l'évidence que la simplicité de cette conception est également sa principale faiblesse.

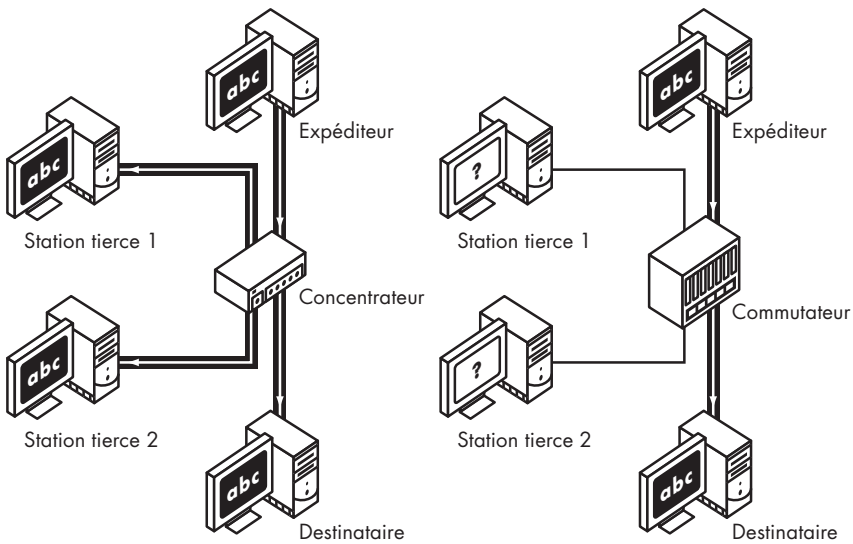


Figure 5.5

Les concentrateurs comparés aux commutateurs dans les réseaux locaux.

Les commutateurs, la génération suivante des concentrateurs, se sont révélés être la solution. Équipés d'un processeur décent et de mémoire, ils représentent une alternative plus coûteuse aux concentrateurs et offrent, dans des circonstances normales, une analyse complémentaire de haut niveau des trames Ethernet. Cette analyse associe les adresses matérielles à certains ports en particulier et optimise le routage des trames en transférant certains paquets directement au port adéquat (en mode unicast), au lieu de les diffuser à l'ensemble des parties (voir Figure 5.5). Cela permet d'améliorer considérablement les performances dans des réseaux plus étendus.

NOTE

Autre fait amusant : les vrais concentrateurs ont quasiment disparu de nos jours. Presque tous les périphériques 10/100 Mo commercialisés actuellement sous le nom de concentrateur utilisent en fait un chipset de commutateur basique. Il est tout simplement moins cher de reconditionner la puce que de développer et de conserver plusieurs variantes.

Je suppose qu'à ce stade vous vous demandez où je peux bien vouloir en venir. Quel rapport entre les modems et la divulgation d'informations ? Quelle signification ont les liaisons série dans ce contexte ? Que viennent faire là les réseaux Ethernet ? Et que sont donc les blinkenlights ?

Merci de poser ces questions. Je suis sur le point de répondre à la dernière.

Les blinkenlights et les communications

Historiquement, presque tous les ordinateurs dont la taille avoisinait celle des réfrigérateurs étaient équipés de nombreuses interfaces de diagnostic bien en vue. Il s'agissait notamment de tableaux de minuscules lumières qui affichaient, entre autres choses, certaines propriétés sur l'état interne de la machine, comme les registres internes ou les flags de l'unité de traitement de base ou qui indiquaient si le chat vivant sous la machine avait été nourri ce jour-là. Les ordinateurs devenant plus fiables et plus compacts et l'utilisateur moyen n'ayant plus à comprendre le fonctionnement interne de la machine pour l'utiliser efficacement, ces lumières ont commencé à disparaître de nombreux appareils. L'augmentation croissante des vitesses d'horloge a également contribué à ce déclin – la plupart du temps, il n'était plus possible pour les humains d'obtenir des renseignements à partir d'un signal visuel qui change des milliers ou des millions de fois par seconde.

Pourtant, les lumières existent toujours dans certaines applications. Par exemple, presque tous les périphériques réseau possèdent des diodes électroluminescentes (DEL) sur leur panneau avant ou arrière. Elles fournissent des diagnostics sur les liaisons, en indiquant si un module ou un socket en particulier fonctionne correctement, si une partie est connectée, si les données sont transférées, et ainsi de suite. Mais ces lumières ne sont pas seulement un outil de diagnostic. Leur clignotement hypnotise et crée une sensation étrange mêlant l'incertitude, la peur et le respect dans le cœur des novices qui entrent dans une salle de serveurs.

Le terme blinkenlights ou blinkenlichten a été utilisé pour décrire l'adoration des DEL de diagnostic sur le matériel informatique depuis ses débuts, à l'époque où le geek informatique baignait dans une lueur verte apaisante durant ses longues nuits solitaires passées devant son écran. Il provient d'une note amusante en pseudo-allemand (lui-même une parodie d'une autre blague de la Seconde Guerre mondiale sans rapport avec

l'informatique) qui était affichée dans les laboratoires d'IBM au cours des années 1950. Cette note se retrouva ensuite dans la majorité des salles de serveurs et dans les laboratoires scientifiques du monde entier. La voici, tirée du dictionnaire du Hacker d'Eric S. Raymond :

```
ACHTUNG!
ALLES LOOKENSPEEPERS!
Alles touristen und non-technischen looken peepers! Das
computermachine ist nicht fuer gefingerpoken und mittengrabben. Ist
easy schnappen der springenwerk, blowenfusen und poppencorken mit
spitzensparken. Ist nicht fuer gewerken bei das dumpkopfen. Das
rubbernecken sichtseeren keepen das cotton-pickenen hans in das
pockets muss; relaxen und watchen das blinkenlichten*.
```

Le matériel de communication est un des derniers domaines dans lesquels les blinkenlights règnent et prospèrent. Mais ce n'est pas tout. Presque tous ces dispositifs utilisent des lignes série pour les communications. Et, dans un souci de simplicité et pour des raisons esthétiques, "l'activité" des diodes est parfois presque directement reliée par câble, par un simple circuit pilote, à la ligne émettrice ou réceptrice de l'appareil.

Les conséquences des diodes esthétiques

Il a fallu des décennies pour découvrir le problème et, quand cela fut fait (en 2002), cela nous a tous semblé tellement évident qu'il ne nous restait qu'à nous frapper la tête sur le clavier.

Joe Lughry et David A. Umphress, dans un document de recherche intitulé "Leakage from Optical Emanations"[4], dévoilaient un nouveau type de divulgation du signal par certains types d'équipements de réseau, le plus souvent des modems. Ils conclurent que l'observation de ces lumières ne se résumait pas au simple plaisir de regarder ces lumières magiques à l'œil nu.

Les DEL, contrairement aux ampoules à incandescence, s'allument et s'éteignent presque instantanément. Ce n'est pas surprenant puisque, après tout, les DEL haut de gamme sont utilisées pour contrôler les liaisons à fibres optiques et d'autres canaux de communication optoélectroniques. À ce titre, le clignotement d'une DEL branchée à une ligne

---

* *NdT :* Ce texte parodique écrit dans un mélange de pseudo-anglais et de pseudo-allemand est difficilement traduisible. La traduction donnée ici est assez libre et n'est livrée qu'à titre indicatif :
AVERTISSEMENT!
À tous les touristes et observateurs qui ne sont pas des techniciens ! Cet ordinateur n'est pas destiné à être tripoté avec vos doigts ! Vous risquez facilement de faire une erreur, de faire sauter les plombs comme du pop-corn et de faire cracher des étincelles à l'ordinateur. Il ne doit pas être utilisé par les imbéciles. Les curieux doivent garder leurs mains caoutchoutées de ramasseurs de coton dans leurs poches ; détendez-vous et contentez-vous d'admirer les lumières qui clignotent.

de transmission de données en série peut en fait souvent représenter chaque bit transmis sur le câble. Si l'on dispose d'un moyen d'enregistrer cette activité à une vitesse suffisante, il devrait être possible de récupérer ces informations, du moins tant que la petite lumière clignotante sur un périphérique est visible à l'œil nu (ou avec un téléobjectif).

Cette étude fut à la fois encensée et méprisée si bien qu'une grande confusion s'ensuivit et que peu de choses changèrent. Ce document entraîna de nombreux rapports conflictuels, mais son postulat de base est simple et vraiment beau. La beauté de cette technique vient du fait qu'il est simple de mettre au point un dispositif pour recevoir le signal : les homologues des DEL, tout aussi bon marché et répandus (les photodiodes et les phototransistors) sont faciles à acquérir et tout aussi faciles à interfacer avec l'ordinateur. Et la zone d'exposition, contrairement à la plupart des activités TEMPEST dont nous avons parlé au Chapitre 3, ne relève pas simplement de la légende urbaine ni de résultats obtenus uniquement en laboratoire mais peut être directement observée et mesurée.

Au cours de leurs recherches, les auteurs ont effectué une série d'expériences afin de vérifier que le signal pouvait être obtenu avec succès depuis une distance de 20 m sans avoir besoin de conditionner des signaux numériques supplémentaires. On peut logiquement en déduire que cette distance peut être largement supérieure, en particulier si l'on utilise des optiques de bonne qualité (les auteurs ont utilisé une focale de 100 mm, un objectif f/2.0 pour leur essai, mais les reflex de milieu de gamme que possèdent de nombreux photographes amateurs disposent d'un téléobjectif bien meilleur (reflex mono-objectif). Et ceux qui sont prêts à dépenser beaucoup d'argent peuvent acheter un objectif de très haute qualité avec une longueur focale allant jusqu'à 1 200 mm).

Le document adopte une position défensive dans plusieurs cas et un lecteur attentif pourrait être tenté de conclure que certains des dispositifs étudiés ne sont pas vulnérables à ce problème. En particulier, certains périphériques Ethernet montrent un type plus subtil de vulnérabilité, comme vous le verrez dans la section consacrée à la prévention, plus loin dans ce chapitre. Mais examinons d'abord le problème avec nos propres yeux (informatisés).

Construire son propre dispositif d'espionnage…

La construction d'un dispositif d'espionnage est si simple qu'il devient très tentant de le faire. Cette section contient plusieurs suggestions et pistes sommaires sur la façon de construire et de connecter un tel dispositif à un simple ordinateur. Bien que le circuit ne soit pas particulièrement complexe et ne nécessite ni de grandes connaissances en soudure ni de logiciel de conception de cartes imprimées, il est souhaitable d'avoir un minimum de maîtrise en électronique et un peu de bon sens. Bien que les interfaces externes des ordinateurs actuels soient assez robustes et fiables, il existe toujours un risque d'endommager son matériel lorsqu'on lui ajoute des dispositifs artisanaux

et innovateurs, dans un bref moment de folie. Cela est arrivé aux meilleurs d'entre nous.

La conception de base est extrêmement simple. Elle ne demande qu'un seul phototransistor (un composant formé d'un transistor dirigé par une photodiode), un transistor NPN (négatif-positif-négatif) de faible puissance pour amplifier un peu plus le signal (ce n'est pas toujours nécessaire) et une série de potentiomètres (peut-être un potentiomètre de 10 k ; il suffit de disposer de suffisamment de flexibilité) pour abaisser à titre expérimental la tension et contrôler la sensibilité du circuit et la tension de seuil. Il n'y a pas d'exigences particulières pour les composants, même si leur coût peut varier en fonction de ceux que vous utilisez. Veillez à sélectionner un phototransistor qui ait une réponse décente dans la gamme de lumières visibles, même si tous les phototransistors bon marché devraient fonctionner (À titre de référence, une DEL verte émet une longueur d'onde d'environ 520 nm).

Un exemple de conception du circuit est illustré à la Figure 5.6.

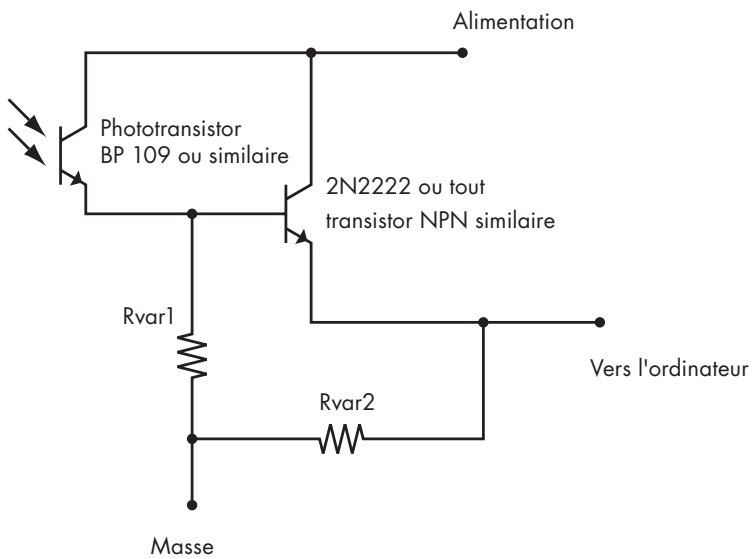


Figure 5.6
Un simple circuit de réception.

Le circuit a une tension de fonctionnement optimale d'environ 5V et un faible courant maximal : une alimentation capable de délivrer 10 à 50 mA est plus que suffisante. Un mot d'avertissement : si vous utilisez une alimentation pouvant produire une tension plus élevée, vous risquez d'endommager le port ou l'ordinateur. Même chose si vous

utilisez une alimentation plus puissante et n'empêchez pas qu'un courant ayant une tension plus élevée circule dans le circuit.

NOTE

Si Rvar1ou Rvar2 sont configurés avec une très faible résistance, cela peut entraîner un court-circuit. Si vous voulez jouer avec les boutons sans souci, il serait peut-être préférable d'ajouter une résistance pour limiter le flux.

Vous devez protéger le phototransistor des sources lumineuses externes, en l'enfermant par exemple dans un tube opaque. Comme le phototransistor n'a aucun mécanisme de focus, il n'est pas susceptible de capter des signaux plus lointains (autres que la lumière ambiante). Ainsi, pour les premiers essais, il est préférable de le couvrir entièrement pour simuler l'obscurité, puis de le présenter à une DEL pour exciter le circuit. Vous pouvez également temporairement connecter une autre DEL entre le GND et la ligne de sortie pour tester le circuit. La DEL de test doit s'allumer lorsque le capteur est dirigé vers une source lumineuse mais doit sinon être assez sombre.

…et l'utiliser avec un ordinateur

Si le circuit comprenant une DEL de test fonctionne, parfait ; vous avez construit une télécommande fantaisiste. Comme les phototransistors génériques et bon marché sont désireux de capter la lumière infrarouge, votre création doit "traduire" les infrarouges (IR) en lumière visible, mais c'est à peu près la seule chose amusante qu'il fera. Pour la rendre un peu plus utile, vous devez interfacer le circuit avec l'ordinateur. Une bonne méthode de le faire consiste à utiliser l'interface de l'imprimante (LPT), si votre ordinateur en possède une. Malheureusement, ce formidable outil matériel pour le pirate informatique tombe en désuétude au profit de conceptions plus compactes et fantaisistes.

Bien que conçu initialement pour être unidirectionnel (pour la sortie uniquement), l'interface LPT renvoie un certain nombre de lignes sur le statut de l'imprimante, comme *paper out* (plus de papier), *busy* (occupé) et *acknowledgement* (acquittement), qui ont pour but de fournir un moyen à l'imprimante de signaler des problèmes. Vous pouvez facilement lire les données qui sortent de cette interface en accédant au port 0x379 (le registre d'état LPT1) sur un système compatible PC. En raccordant le circuit à un port parallèle, vous pouvez facilement transmettre des informations à l'ordinateur. Même si vous préféreriez sans doute connecter le circuit à une interface différente, LPT est beaucoup plus rapide que RS-232, par exemple, et vous n'aurez pas à vous soucier de protocoles mondains, de systèmes de signalisation ou de niveaux de tension inhabituels. De plus, contrairement à USB et à quelques autres solutions actuelles, vous n'avez pas

besoin de contrôleurs spéciaux pour implémenter un protocole assez complexe pour être en mesure de parler à votre PC.

NOTE

Bien que LPT propose également des modes de fonctionnement bidirectionnels (ECP ou EPP), il est généralement inutile d'essayer d'utiliser cette fonctionnalité pour une tâche aussi simple. Dans le mode unidirectionnel, quatre bits sont disponibles en entrée, ce qui est plus que suffisant pour cette application ; le passage à des modes bidirectionnels comme EPP ou ESP fournit un quatrième bit supplémentaire.

C'est à vous de choisir quelle ligne d'état utiliser. Le Tableau 5.3 montre le brochage du connecteur DB25 utilisé sur un port d'imprimante. Les lignes surlignées en gris peuvent être utilisées pour l'entrée.

Pour interfacer le circuit avec ce port, vous pouvez simplement connecter le point de référence de la terre sur le connecteur avec celui utilisé dans votre circuit et ensuite brancher la ligne de sortie sur une des cinq broches (n'oubliez pas de débrancher auparavant la DEL utilisée pour les diagnostics). Ensuite, surveillez l'état du port lorsque vous l'exposez à la lumière puis que vous couvrez le capteur. Dans les deux cas, la valeur lue dépend de la façon dont vous avez branché le circuit. La valeur exacte n'a pas d'importance, tant que les deux valeurs sont différentes.

Tableau 5.3 : Le brochage LPT

Port LPT : Brochage DB25 (mode standard)		
Broche	***Nom***	***Fonction***
1	Lumière stroboscopique	Contrôle sortie bit 0
2	D0	Données sortie bit 0
3	D1	Données sortie bit 1
4	D2	Données sortie bit 2
5	D3	Données sortie bit 3
6	D4	Données sortie bit 4
7	D5	Données sortie bit 5
8	D6	Données sortie bit 6
9	D7	Données sortie bit 7

Tableau 5.3 : Le brochage LPT *(suite)*

Port LPT : Brochage DB25 (mode standard)		
Broche	***Nom***	***Fonction***
10	$\overline{\text{ACK}}$	État entrée bit 2
11	Busy	État entrée bit 3
12	Paper out	État entrée bit 1
13	Select in	État entrée bit 0
14	$\overline{\text{Autofeed}}$	Contrôle sortie bit 1
15	$\overline{\text{Error}}$	État entrée (inutilisé)
16	$\overline{\text{Init}}$	Contrôle sortie bit 2
17	$\overline{\text{Select}}$	Contrôle sortie bit 3
18	GND	Masse (0V)
19	GND	Masse (0V)
20	GND	Masse (0V)
21	GND	Masse (0V)
22	GND	Masse (0V)
23	GND	Masse (0V)
24	GND	Masse (0V)
25	GND	Masse (0V)

Puisque la logique de la puce demande des niveaux d'entrée quelque peu différents de ceux de votre diode DEL de test, vous pourriez devoir régler `Rvar2` jusqu'à ce que vous obteniez des lectures distinctes du port quand vous couvrez la sonde et quand vous l'exposez à la lumière. Pour accomplir ceci, il est préférable de pouvoir surveiller le port en temps réel sur l'ordinateur lui-même.

La manière dont vous pouvez contrôler l'état du port dépend du système d'exploitation et du langage de programmation que vous utilisez. En langage C, la fonction utilisée pour lire la valeur de chaque port est `inb(port)`. Donc, dans ce cas particulier, vous utiliserez `inb(0x379)` et vérifierez la valeur en retour. Dans d'autres langages, le nom de la fonction est susceptible d'être similaire (essayez de rechercher `in`, `inport`,

`readport`, etc.). Les utilisateurs de Windows peuvent également trouver l'utilitaire intégré de "débogage" et sa fonction "i" (*port read*) très pratique.

NOTE

Sur certains systèmes, comme Linux, vous aurez peut-être tout d'abord besoin que le système vous donne l'autorisation d'accéder à un port spécifique. Consultez la documentation de `iopl(3)` ou d'un appel similaire pour plus d'informations.

À ce stade, vous êtes prêt à commencer. Vous pouvez choisir de pointer la sonde vers n'importe quelle DEL d'un périphérique, régler le capteur en fonction de sa luminosité et commencer à lire les motifs alternatifs des signaux lumineux pour découvrir éventuellement à quelles informations échangées ces signaux correspondent.

NOTE

Si vous êtes curieux, vous pouvez essayer d'examiner la luminosité de la diode et pas seulement une représentation binaire de son état. Il pourrait s'avérer que, même si une DEL en particulier n'est pas conçue pour établir directement un lien entre un signal sur la ligne série et ses modes de clignotement, il y a un certain dialogue croisé analogique entre les circuits, si bien que le signal de la ligne série a une certaine influence sur la luminosité. Un convertisseur analogique-numérique bon marché comme le TLV571 de Texas Instruments demande juste à être utilisé de cette manière.

Vous pouvez utiliser cette approche pour échantillonner une fréquence de moins de 1 million de bits par seconde, ce qui devrait suffire pour capter les transmissions sur un grand nombre d'interfaces, mais pas nécessairement sur les ports Ethernet (qui transmettent au moins 10 millions de bits par seconde). Au-delà de cette capacité de capture, le port LPT atteindra probablement la limite de son débit physique, mais il ne faut pas désespérer : tant que le capteur (phototransistor) peut clignoter à un taux suffisant pour capter les communications en question, vous avez toujours une option. N'oubliez pas que LPT est un port parallèle. Pour atteindre des vitesses plus rapides de capture, comme celui nécessaire pour Ethernet, utilisez une simple horloge, un compteur et un ensemble de verrous échantillonneur-bloqueur (comme 74LS377) afin de stocker les données de façon séquentielle entre les tentatives de lecture du port côté ordinateur. Vous pouvez accumuler ces informations pendant une courte période de temps, puis, en utilisant plus d'une seule broche d'état (ou en commutant le port en mode bidirectionnel), envoyer facilement plusieurs bits (échantillons) à l'ordinateur, en une seule fois, dans un seul cycle de lecture, améliorant ainsi le taux de lecture par quatre ou huit.

Je vous épargnerai une autre excursion, peut-être inutile, dans le monde de l'électronique. Si l'idée d'un échantillonnage analogique ou à grande vitesse vous intéresse ou si

vous voulez simplement vous amuser à rassembler divers éléments et les brancher à un ordinateur, vous pouvez jeter un œil sur mon didacticiel assez complet qui se cache sous un projet de création d'un robot contrôlé par ordinateur. Vous devriez pouvoir trouver ce didacticiel à l'adresse suivante : http://lcamtuf.coredump.cx/robot.txt.

Et, maintenant, si vous êtes plus intéressé par la sécurité dans la pratique, la section suivante aborde brièvement la façon de se protéger de ce problème, sans pour autant couvrir toutes les DEL du bureau avec du ruban adhésif.

Empêcher que les DEL ne divulguent des données (et pourquoi cela ne fonctionne pas)

La solution la plus simple à ce problème, suggérée d'ailleurs dans l'étude originale, consiste à recourir à la *modulation de l'impulsion*, autrement dit à prolonger le clignotement de certaines diodes sur un indicateur. Normalement, il est alors impossible de récupérer des données utilisables. Les circuits de *modulation de l'impulsion* sont un groupe de dispositifs assez simples qui prolongent la durée d'un signal d'entrée "élevé" pendant un certain temps. Les systèmes les plus simples reposent sur une capacité qui se charge en présence d'un signal d'entrée puis se décharge lentement. Cette capacité est reliée à un *discriminateur binaire*, un dispositif qui convertit les données analogiques en sortie binaire en appliquant un certain seuil (1 logique en sortie pour toutes les tensions d'entrée supérieures à *n* et 0 pour toutes les tensions d'entrée inférieures). Dans ce cas, il utilise un certain niveau de charge de condensateur comme point de séparation.

Des conceptions plus fiables et plus avancées, y compris des circuits entièrement numériques, sont également fréquentes et toutes peuvent être utilisées dans les concentrateurs et les commutateurs pour rendre l'observation des DEL agréable. Sans elles, nous aurions l'impression que les DEL, qui clignotent plus de cinquante fois par seconde (ce qui est considéré comme la limite de notre capacité à percevoir un scintillement), produisent en fait une lumière trouble mais constante. Un discriminateur force la DEL à avoir plus souvent une valeur de 1 que de 0, en prolongeant la durée de chacune des impulsions égales à 1. La diode est alors plus lumineuse et clignote moins souvent. La Figure 5.7 montre le comportement d'une telle modulation de l'impulsion : une seule crête (une seule valeur 1) est étendue pour durer trois fois plus longtemps, tandis que tous les 0 sont laissés tels quels.

Bien que le but premier soit esthétique, comme je l'ai dit, cela semble également être un bon moyen de résoudre le problème de la divulgation de l'information par les émissions lumineuses, puisque l'attaquant ne peut plus déduire que certaines propriétés générales

du trafic. Autrement dit, il peut au mieux savoir à quel moment quelque chose est émis ou non*.

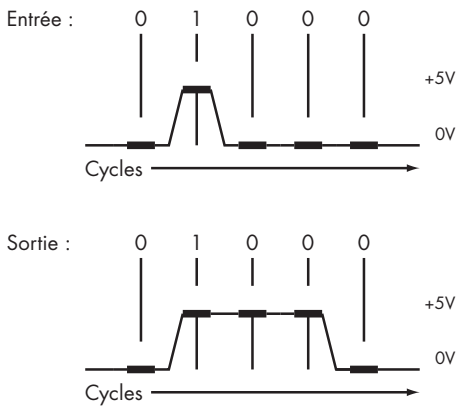


Figure 5.7

Comportement d'un modulateur d'impulsion, 3x.

Toutefois, ce qui semble être une bonne solution ne l'est pas toujours. Prenons l'échantillon suivant de données et le signal correspondant de la ligne série :

Données :

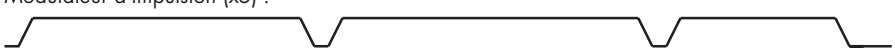
0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0

Signal NRZ :

Imaginons que le signal est traité par un modulateur d'impulsion qui multiplie par 5 la durée de tous les 1 pendant cinq cycles supplémentaires (les auteurs de l'étude originale proposent une limite de sécurité de 2x, mais nous l'exagérons pour être plus clairs).

Signal original :

Modulateur d'impulsion (x5) :



Données qui peuvent être lues après modulation de l'impulsion de la DEL :

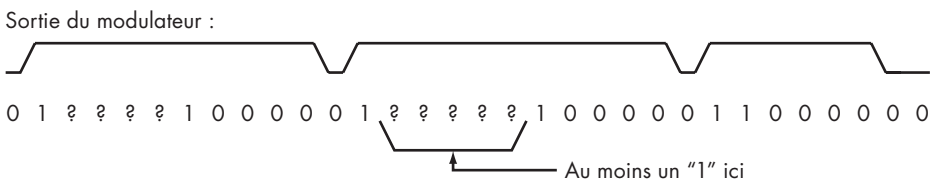
0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0

* Techniquement parlant, cela est encore le lieu d'une attaque, comme nous l'avons vu au Chapitre 1, mais est beaucoup moins efficace et pratique, car nous n'obtenons qu'une idée de ce qui se passe, et non une copie des données.

Même s'il semble que presque toutes les informations importantes du signal d'entrée que nous voulons intercepter ont été perdues, il est possible d'en récupérer une bonne partie en faisant quatre observations importantes :

- De toute évidence, toutes les zones où la production du modulateur est égale à zéro correspondent à une valeur zéro dans le signal original.
- Chaque augmentation de la durée des 1 doit avoir été déclenchée par la présence d'un 1 au départ dans le flux original.
- Chaque exécution de L 1 doit avoir à l'origine contenu au moins un 1 pour tous les *N* cycles, avec *N* comme facteur d'étirement pour ce circuit, car sinon l'exécution comprendrait des vides. Le compte des 1 dans un bloc de données représentées par un seul 1 étiré en sortie est supérieur ou égal à L/N.
- Chaque exécution se termine après exactement *N*-1 zéros dans le flux original. Nous savons que ces zéros doivent avoir été précédés par 1, car sinon l'exécution aurait pris fin plus tôt.

En appliquant ces connaissances à l'exemple précédent, nous pouvons reconstruire la plus grande partie des données d'origine, de la façon suivante :



Dans le précédent exemple, assez réaliste, moins de 9 bits de données sur 32 ont été perdus par la modulation de l'impulsion et ne peuvent pas être reconstruits de façon concluante (indiqués par des points d'interrogation dans le schéma). Ainsi, nous avons récupéré 99,999988 % du champ de recherche. Nous devons deviner les données restantes, ce qui (en particulier si les données espionnées constituent un texte en langue française, comme un courrier électronique) est assez simple à effectuer comparé au point de départ. Les auteurs de l'étude pensent qu'un temps de modulation de l'impulsion N = 1,5 ou N = 2 est suffisant pour masquer les données, mais ce n'est pas nécessairement le cas.

Le précédent programme de reconstruction fonctionne lorsque la modulation porte sur des 0 ou des 1. Certaines liaisons utilisent le codage RZ (retour à zéro) comme le codage Manchester mentionné plus tôt. Dans ce cas, comme le signal alterne constamment, la modulation 2x pourrait en effet être suffisante pour masquer toutes les données. Toutefois, cela n'est vrai que si la DEL est pilotée par un signal précédant le premier décodage NRZ interne, ce qui n'est pas le cas dans la plupart des cas. En fait, il est souvent stupide

d'appliquer une modulation des impulsions sur un signal qui utilise le codage RZ au sens où la DEL serait constamment allumée. En premier lieu, il ne semble y avoir donc aucun intérêt à le faire.

Comme indiqué précédemment, la qualité du modulateur d'impulsions et sa sensibilité aux interférences produites par les autres circuits internes posent un problème supplémentaire : les fluctuations de tension de la DEL qui se traduisent par de légères variations de la luminosité lors d'une période "modulée" pourraient divulguer certaines informations. C'est en particulier le cas des solutions qui reposent sur l'utilisation de condensateurs.

Ainsi, certains systèmes, en particulier les périphériques Ethernet connus pour déployer une modulation des impulsions, peuvent être en partie vulnérables à une attaque. Et ce même si l'étude originale dont nous avons parlé précédemment conclut, en se fondant sur l'observation d'un enregistrement du schéma de clignotement avec un oscilloscope, qu'il n'existe pas de corrélation directe entre les données transmises et le comportement d'une DEL.

La solution optimale, en particulier avec d'autres types de codages ou lorsque la modulation de l'impulsion n'est pas souhaitable pour d'autres raisons (si par exemple le concepteur veut éviter que la lumière de la DEL n'apparaisse constamment pendant la transmission), consiste à échantillonner la ligne à une fréquence relativement faible (20 Hz, par exemple) et à la verrouiller à un registre qui la conserve jusqu'à l'échantillon suivant et qui contrôle également la DEL.

Revenons maintenant au langage courant.

Matière à réflexion

Il existe beaucoup d'autres cas tout aussi intéressants de fuite d'informations par les émissions lumineuses que celles des DEL des périphériques réseau, même si la quantité d'informations divulguées peut être nettement plus faible. Prenez par exemple les DEL indiquant l'activité des disques. Bien sûr, la communication des disques ne fait pas appel à des signaux en série mais une partie des données, allant de 1 octet à des mots de 32 bits, sont envoyées simultanément en utilisant un ensemble de lignes de signaux. Et, bien que la DEL ne serve généralement à indiquer que l'état d'une ligne de contrôle spécifique, il est encore possible de déduire de nombreux aspects de l'activité du système en mesurant les temps de recherche ou la quantité de données stockées et lues (selon à quoi la DEL est réellement attachée, il peut être possible de mesurer l'un ou l'autre, voire les deux). Bien qu'il soit peu probable que ces informations procurent à l'attaquant un quelconque avantage immédiat, certaines déductions effectuées à partir des activités d'entrée-sortie combinées à l'observation de la DEL du disque dur

pourraient permettre de tirer des conclusions intéressantes, même si je ne suis au courant d'aucune recherche dans ce domaine.

De nombreux périphériques USB et d'autres interfaces propriétaires peuvent également représenter des cibles potentielles. Comme mentionné précédemment, l'USB est un bus série, et certains périphériques USB ont des indicateurs d'activité.

Diverses autres recherches inhabituelles et mystérieuses sur la divulgation d'informations ont également fait l'objet de recherches partielles ou été envisagées. Il s'agit notamment de mesurer les effets acoustiques de la recharge des condensateurs, puisque le processeur consomme différents niveaux de puissance en fonction des instructions qu'il exécute[5], et de réaliser une analyse statistique de la consommation d'énergie d'une boîte noire[6]. Là encore, aucune recherche réellement exhaustive n'a été menée sur d'autres canaux de divulgation d'information que celles effectuées sur les émanations des champs électromagnétiques, et cela semble être une bonne idée à étudier. Bonne chance.

6

Échos du passé

Dans lequel, à l'exemple d'une faille curieuse d'Ethernet, on apprend que mieux vaut s'exprimer avec précision.

Le chapitre précédent abordait les notions élémentaires de la communication Ethernet. Ce mécanisme apparemment infaillible et étonnamment simple semble être incapable d'engendrer de sérieux problèmes de sécurité, à l'exception possible d'un abus de confiance provoqué par la diffusion régulière de données à tous les membres du réseau. Cette propriété des réseaux Ethernet est bien connue et bien comprise, et de nombreux palliatifs existent, comme les commutateurs, les ponts et la segmentation du réseau, pour n'en citer que quelques-uns.

Néanmoins, ce problème se manifeste de manière totalement imprévue, en raison le plus souvent d'un mauvais choix de mots ou de l'absence de ce choix, dans les exigences d'implémentation officielles des pilotes Ethernet. Ce problème d'implémentation est si répandu que je lui consacre ce chapitre. Il fournit une étude de cas intéressante des problèmes dont personne n'est responsable.

Construire la tour de Babel

Le protocole Ethernet fournit un moyen simple de diffuser des octets sur un câble : un système de codage des données de bas niveau et un format de données pour contenir une partie de l'information. La trame Ethernet contient des informations sur la disposition locale des données qu'elle véhicule (autrement dit qui la transmet et qui doit être le bénéficiaire) ainsi qu'une brève description du type d'informations encapsulées. Des méthodes complémentaires de détection des erreurs sont également fournies, puis l'ensemble de la trame est envoyé à un destinataire potentiel et à tous les autres systèmes. En termes de fonctionnalités, Ethernet est semblable aux systèmes d'encapsulation de portion de données utilisées sur des supports différents ou dans différentes applications, comme le relayage de trame (*frame relay*), le mode de transfert asynchrone (ATM), le protocole point à point (PPP), et ainsi de suite.

La question suivante se pose : "Quelles données devraient être véhiculées par telle trame Ethernet ?" Les ordinateurs utilisent des centaines de formats et de protocoles et peuvent exécuter des applications allant de la simulation scientifique au jeu en passant par les chats. Donc, même s'il est possible de simplement encapsuler les données destinées à un hôte distant au sein d'une trame Ethernet, cela constitue généralement une mauvaise solution car le destinataire ne saura pas comment les traiter. Est-ce un courrier électronique ? Une photo ? Ou peut-être des données de configuration ? Impossible de le savoir. En outre, comme un ordinateur exécute généralement tout un ensemble de programmes quasi simultanément, la distinction devient encore plus floue.

À plus grande échelle, Ethernet pose encore un autre problème ; comment atteindre l'autre terminal ? La diffusion des données à tous les membres est aisée sur un réseau local, mais que se passe-t-il si l'autre système, celui qu'un des utilisateurs locaux espère joindre, n'est pas sur le réseau local ? Que se passe-t-il s'il se trouve sur un réseau étendu (WAN) et utilise un protocole de liaison complètement différent ? Même si une solution peut être trouvée pour acheminer le trafic jusqu'à destination, une question plus fondamentale subsiste : comment traiter le paquet ?

Ethernet utilise ses propres systèmes d'adressage spécialisés. Il appelle les hôtes par le numéro d'identification matériel en théorie unique (l'adresse MAC pour *Media Access Control*) que le constructeur intègre à chaque carte Ethernet. Ces chiffres n'ont de signification que pour Ethernet ; ils n'ont aucun sens pour les autres types de réseaux et sont presque impossibles à utiliser pour retrouver un élément matériel si on ne fait pas partie de la configuration locale. Cela soulève une question de confiance. Par exemple, qui a acheté la carte avec l'adresse 00:0D:56:E3:FB:E4 et où se trouve-t-il maintenant ? Peut-on penser en toute confiance qu'il s'agit vraiment de l'acheteur de cette carte et non d'un imposteur ?

Les systèmes de bas niveau d'adressage des hôtes comme celui-ci ne sont généralement pas d'un grand secours pour relayer les données à destination, sauf si le matériel ayant une adresse MAC particulière est relié directement au réseau physique de l'expéditeur.

Il n'existe aucun moyen direct d'établir un lien entre l'identifiant d'un périphérique physique et un emplacement particulier sur le globe ni de déterminer quel chemin utiliser pour lui envoyer des informations.

Le modèle OSI

Les protocoles réseau ont été conçus pour favoriser la communication entre les nœuds locaux ou, dans certains cas extrêmes, entre deux points fixes sur une liaison partagée. Pour rendre l'interconnexion possible et certaines utilisations pratiques des réseaux réalisables, une structure hiérarchique des protocoles réseau appelée modèle d'interconnexion en réseau des systèmes ouverts (OSI, pour *Open System Interconnection*) a été conçue.

Le modèle OSI (voir Figure 6.1) définit le niveau de connexion physique comme la première couche et lui attribue les fonctionnalités de plus haut niveau. Les protocoles de liaison forment la deuxième couche (la couche de liaison des données) et sont définis comme une façon de communiquer avec d'autres nœuds locaux qui utilisent la même liaison physique. Ces protocoles transportent des données de plus haut niveau indépendantes du protocole de liaison et définies comme la troisième couche (la couche réseau) dans le modèle OSI. Le protocole Internet (IP) est le plus important exemple de ce protocole.

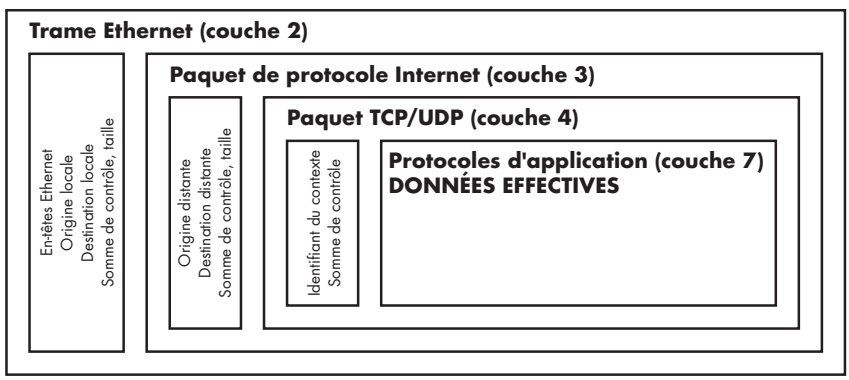


Figure 6.1

Un exemple de disposition des données physiques dans le modèle OSI.

La troisième couche est conçue pour fournir des informations sur la disposition générale du trafic ainsi que l'identification universelle de l'origine et de la destination finale des données à l'aide de l'adressage spécifique du réseau, ce qui rend plus facile l'acheminement du paquet. Contrairement aux protocoles de la deuxième couche, la troisième couche n'est pas écartée ou modifiée en cours de route et elle est dépourvue de toutes fonctionnalités spécifiques à la liaison, comme les adresses MAC, la surcharge CSMA/CD (*Carrier Sense Multiple Access* avec détection des collisions), et ainsi de suite.

La quatrième couche fournit les moyens d'établir des canaux de communication spécifiques de bout en bout sur une machine donnée. Cela permet la communication simultanée de plusieurs types et canaux. Aucun des protocoles de quatrième niveau n'a besoin d'être compris par les systèmes intermédiaires pour acheminer correctement les données à destination. Les paquets ne sont interprétés que par le destinataire final pour déterminer quelle application doit recevoir les données et quel lien cette information a avec les paquets adjacents.

Les couches suivantes du modèle OSI sont peut-être moins intéressantes et ont tendance à se confondre. Le cinquième niveau est censé fournir des caractéristiques de fiabilité qui sont souvent intégrées dans les protocoles de quatrième niveau, comme TCP/IP (*Transmission Control Protocol/Internet Protocol*) ou au niveau de l'application. Dans certains cas, ils ne sont même pas implémentés du tout s'il n'est pas nécessaire de garantir des communications fiables. Le sixième niveau fournit des fonctions "bibliothèques" pour la décompression et le décodage des données. Comme le cinquième niveau, il est généralement perçu en termes de fonctionnalité au niveau de l'application. Enfin, la septième couche, la couche application, est le lieu où les données sont transférées dans un format précis.

Notez que les couches plus élevées dans le modèle OSI sont indépendantes des couches inférieures puisqu'elles s'appliquent aux données transportées. Lorsque le moment est venu, les couches inférieures peuvent être progressivement éliminées sans perdre de données ou la capacité à poursuivre leur traitement. La deuxième couche est supprimée par chaque système intermédiaire, la troisième couche peut l'être une fois que les données sont fournies au système de destination. La quatrième couche est abandonnée avant de livrer les données à l'application cliente.

La troisième couche reste habituellement complètement indépendante du protocole de liaison sous-jacent et fournit des informations complètes sur l'expéditeur et le destinataire, un mécanisme de protection de l'intégrité (somme de contrôle) et des informations sur la taille de la charge transmise. C'est précisément ce que fait le protocole IP.

Une conséquence importante de cette conception est que toutes les informations superflues annexées au paquet sur la deuxième couche au cours du transfert n'ont aucune incidence sur la façon dont les informations IP sont interprétées par le destinataire.

La phrase manquante

Au chapitre précédent, au cours du débat sur la conception d'Ethernet, j'ai mentionné une obligation intéressante qui découle de la nécessité de fournir et de propager un brouillage du signal fiable pour notifier une collision : la limite de taille *minimale* pour une trame Ethernet.

Cette nécessité existe également dans les spécifications officielles d'encapsulation IP sur Ethernet comme RFC 1042, "une norme pour le transport des datagrammes IP sur les réseaux IEEE 802"[1] qui exige que les trames plus courtes que cette longueur minimale soient bourrées. Ce bourrage peut être réalisé à volonté et n'a aucun effet sur les données transférées sur la couche IP, puisque la longueur du paquet spécifiée dans les en-têtes IP ne change pas. Ainsi, le bourrage ne sera pas interprété par le bénéficiaire comme faisant partie d'un niveau supérieur dans le modèle OSI.

Toutefois, il y a un léger problème. Bien que la norme RFC exige que le rembourrage soit initialisé à zéro, il ne précise pas qui doit le préparer et le fournir ni à quel stade de l'application il doit se produire. Le fait que le bourrage ait une valeur particulière est aussi une exigence assez arbitraire par nature. Par conséquent, aucune attention particulière ne lui est accordée, puisque la manière dont ce bourrage est créé n'aura aucun effet sur le fonctionnement du protocole étant donné que les données superflues sont simplement éliminées à la réception.

Pour ajouter à la confusion, de nombreuses cartes d'interface réseau disposent d'une fonction de bourrage automatique lorsqu'un paquet que le système d'exploitation envoie au matériel est trop court, ce qui n'est évidemment pas effectué pour garantir le contenu spécifique du bourrage si la taille de la trame a déjà été prise en charge dans le logiciel. Cela a entraîné une certaine confusion parmi de nombreux développeurs, qui ont alors choisi d'obéir à l'exigence de taille et d'étendre la taille d'un paquet dans les logiciels en augmentant tout simplement sa longueur déclarée. Ils ne se rendaient souvent pas compte que les données entre la fin du paquet IP et la fin de la trame rembourrée n'étaient pas préparées (initialisées à zéro) par le pilote, le système d'exploitation ou le matériel.

Ce problème est resté largement ignoré pendant des années, même si cela provoquait régulièrement des problèmes réseau qui rendaient fous certains responsables de la sécurité. Les paquets qu'ils recevaient des systèmes locaux contenaient souvent des données inutiles à la fin, comme des fragments du contenu d'un site Web ou même des discussions qui étaient manifestement sans pertinence. Ils accusaient le destinataire (matériel défectueux, logiciel d'analyse du trafic réseau, bibliothèques) mais cessèrent finalement d'en chercher la cause puisque la question avait une importance mineure. Cette question n'a jamais reçu l'attention qu'elle mérite.

Du moins jusqu'en 2003, lorsque Ofir Arkin et Josh Anderson, de @Stake, décidèrent de l'examiner plus attentivement dans leur étude "EtherLeak – Ethernet Frame Padding Information Leaks"[2]. Ils réalisèrent que de nombreux systèmes couramment utilisés, comme Linux, NetBSD, Microsoft Windows et autres, ne parvenaient pas à initialiser la mémoire à la fin de la trame Ethernet après avoir modifié sa longueur. Certaines implémentations échouaient même à changer correctement la taille d'une trame ou à envoyer le bon nombre d'octets à la couche matérielle.

En conséquence, le paquet IP est rempli avec les données stockées dans la partie de la mémoire que le système a précédemment utilisée à d'autres fins. La mémoire peut contenir une partie d'un paquet envoyé précédemment ou un autre fragment du noyau en mémoire, en fonction de la conception du pilote ou du système d'exploitation. Cela, bien sûr, ouvre des possibilités fascinantes en terme de divulgation de l'information : un attaquant envoie discrètement des données légitimes à la victime et, avec un peu de chance, obtient des informations potentiellement sensibles. La quantité d'informations divulguées est généralement suffisante pour justifier qu'on s'en préoccupe.

L'exposition se limite à un seul réseau Ethernet et, à ce titre, est assez localisée et non critique dans un environnement LAN typique. Cela revêt néanmoins une certaine importance et, même si tout réseau local est en partie vulnérable à l'espionnage, ce problème particulier amène à tirer certaines conclusions qui ne sont pas obligatoirement les plus évidentes :

- Sur les systèmes qui utilisent des mémoires tampon dynamiques pour les trames Ethernet sortantes (Linux, par exemple), le remplissage peut divulguer non seulement la trame précédente mais également d'autres parties du contenu de la mémoire, comme les documents édités ou consultés, les URL, les mots de passe ou d'autres ressources sensibles. Dans ce cas, un observateur attentif pourrait être en mesure d'accéder à des informations qu'il ne pourrait pas intercepter autrement sur le réseau.
- Sur les systèmes qui n'utilisent que des mémoires tampon statiques pour préparer les trames Ethernet, cette faille peut être exploitée pour mettre en échec les systèmes de protection contre l'écoute du trafic comme les commutateurs, ce qui permet à l'attaquant d'intercepter les données d'une autre connexion.
- Dans certaines conceptions de mémoires tampon statiques, sur une machine qui possède une interface réseau connectée à un réseau LAN général et une autre interface branchée sur un réseau restreint, des informations d'un autre segment peuvent être exposées et donc des portions de données secrètes, relayées à l'infrastructure publique.

Les auteurs de cette étude ont examiné en détail plusieurs applications open source et ont noté qu'une grande variété d'approches et de mémoires tampon étaient couramment utilisées. Il n'existe pas de système prédominant pour l'allocation et l'utilisation de la mémoire tampon. Leur conclusion ? Un environnement réseau typique diversifié est susceptible d'être affecté par les trois types de problèmes à un moment ou à un autre.

Matière à réflexion

Les questions examinées ici ne sont pas propres à Ethernet ou à la conception d'un réseau. Ces problèmes se posent presque toujours lorsque des directives d'implémentation par ailleurs bien détaillées omettent ou ne traitent que vaguement une étape nécessaire, poussant ainsi de nombreux développeurs à ignorer tout simplement le problème lorsqu'ils implémentent cette norme. S'ils ne disposaient que d'instructions d'ensemble plus vagues, les développeurs seraient probablement obligés de penser à ce problème. Au lieu de cela, ils effectuent l'implémentation pas à pas et risquent beaucoup plus de commettre des erreurs. Les instructions "infaillibles" qui disent comment accomplir certaines tâches plutôt qu'expliquer le but à atteindre échouent souvent à remplir leur rôle.

Nous reviendrons dans la partie III de ce livre sur les problèmes de fuite de protocoles, quoique dans un contexte légèrement différent.

7

La sécurité dans les réseaux commutés

Ou pourquoi les réseaux Ethernet ne peuvent pas vraiment être sécurisés, quels que soient les efforts fournis.

Les réseaux Ethernet ne fournissent pas de moyen simple et universel pour assurer l'intégrité ou la confidentialité des données qu'ils transmettent. Ils ne sont pas non plus conçus pour résister au trafic malveillant, injecté intentionnellement. Ethernet est seulement un moyen d'interfacer un certain nombre de systèmes locaux et considérés comme dignes de confiance.

En théorie, ce niveau de confiance est pratique au stade de la conception et suffisant pour un réseau de systèmes pairs situés à peu près au même endroit. Mais, comme le dit le vieil adage, il n'y a qu'en théorie qu'il n'y a pas de différence entre la théorie et la pratique. Dans la pratique, il y a une différence.

En fait, les réseaux locaux sont difficiles à contrôler totalement et doivent être protégés aussi bien de leurs propres utilisateurs que des menaces extérieures. Tout réseau local qui s'étend est amené à rencontrer un utilisateur malveillant, qu'il soit membre de l'organisation ou non, qui exploitera une faille dans l'un des systèmes.

Cette rencontre n'est qu'une question de temps, comme presque tous les administrateurs réseau l'apprennent un jour.

En pratique, la sécurité des réseaux est l'art de détecter les incidents, de minimiser l'exposition, d'évaluer et de comprendre les risques à tous les niveaux. Cela ne consiste pas seulement à établir un périmètre de défense. Le problème ? Une infrastructure Ethernet est sujette à toutes les formes d'interception des données (détournement, usurpation d'identité). Une fois qu'un intrus ou un utilisateur légitime mais malveillant contrôle un seul système sur le réseau (en franchissant une seule ligne de défense), il peut créer des ravages dans l'infrastructure et avoir accès à certains services et ressources ou en prendre le contrôle avec un minimum d'effort.

Un peu de théorie

Les commutateurs Ethernet, une catégorie de périphériques intelligents destinés à acheminer le trafic unicast de la deuxième couche OSI au port approprié plutôt que le diffuser à tous les nœuds (comme c'est le cas avec les concentrateurs ou les connexions directes), semblent à même de résoudre ce problème. On considère souvent qu'ils résolvent les problèmes de sécurité liés à la capacité d'un système à observer ou à détourner le trafic de systèmes tiers, mais ce n'est pas le cas. La solution n'est pas si simple et cette supposition provoque parfois plus de mal que de bien. Mais reprenons les choses dans l'ordre. Pour comprendre l'exposition, regardons comment les commutateurs Ethernet fonctionnent effectivement.

La résolution de l'adresse et la commutation

Toutes les communications au sein d'un réseau local se fondent sur le mécanisme d'adressage évoqué au Chapitre 5. Des identifiants uniques assignés par le fabricant du matériel à un périphérique d'extrémité en particulier sont utilisés pour contacter les systèmes et fournir les trames de données. Toutefois, la conception d'Internet et de la plupart des réseaux privés actuels repose sur un ensemble universel et plus flexible de protocoles et utilise sur la troisième couche OSI un schéma d'adressage appelé généralement adresse IP (*Internet Protocol*). L'adresse IP est d'abord utilisée pour diriger le trafic mondial vers le réseau local adéquat en utilisant une hiérarchie de tables de routage sur des systèmes intermédiaires situés dans le monde entier ; ce n'est que lorsque le paquet atteint le périmètre du réseau de destination que le destinataire final est localisé par l'ancienne méthode, autrement dit en recherchant son adresse matérielle.

Chaque fois qu'un système sur le réseau local décide de trouver une autre partie locale à l'aide de son adresse IP, il utilise un protocole de résolution d'adresse (*Address Resolution*

Protocol, ARP) pour déterminer l'association entre l'adresse physique de la carte (la base pour contacter les systèmes sur un réseau local) et l'adresse IP, un identifiant universel du système d'interconnexion[1]. L'expéditeur envoie une requête ARP à une adresse broadcast sur le réseau local. Cette adresse réservée est assurée d'être reçue et traitée par tous les systèmes du réseau, indépendamment de l'adresse matérielle réelle affectée à chacun des nœuds. Dans ce scénario, le système qui s'estime en droit d'utiliser l'adresse IP spécifiée dans la requête envoie une réponse à l'expéditeur et divulgue ainsi son adresse matérielle en réponse à la requête. Tous les autres systèmes sont censés ignorer en silence la diffusion du paquet ARP. Après cet échange, les deux parties connaissent l'adresse IP ainsi que les adresses MAC (*Media Access control*) de chacune d'entre elles. Elles doivent mettre en mémoire cache leur découverte dans une zone tampon spéciale afin de ne pas avoir à effectuer d'autres recherches chaque fois qu'une partie des données est échangée, puis elles procèdent réellement à la communication, mais elles sont de plus prêtes à échanger des paquets en se fondant sur l'adressage IP. Cette conception est un exemple merveilleux et charmant de la confiance et de la courtoisie qui existait à une autre époque. Mais comment contenir l'exposition causée par une machine témoin malveillante sur le même réseau qui prétend être quelqu'un d'autre et que faire pour empêcher les utilisateurs les plus curieux ou les plus malveillants d'aller trop loin ? Les fabricants de matériel Ethernet n'aident vraiment pas les administrateurs réseau en rendant possible et facile de changer les adresses MAC sur la plupart des périphériques actuels – pour permettre sans doute à l'utilisateur de les reprogrammer au cas où un lot de cartes se révélerait avoir des adresses en double.

Là encore, les commutateurs semblent résoudre le problème. Le concept de base d'un dispositif de commutation intelligente repose sur la duplication du cache de l'adresse MAC au niveau d'un périphérique réseau intérimaire. Un commutateur est équipé de plusieurs ports Ethernet, chacun relié à un système unique (ou, moins souvent, à un ensemble de systèmes). Mais, plutôt que servir de simples répéteurs et envoyer l'ensemble du trafic reçu sur un port à tous les autres (comme le font les concentrateurs Ethernet), les commutateurs tentent de mémoriser les adresses MAC associées à une machine connectée à chaque port, en créant effectivement une association entre l'adresse MAC et le port, par opposition à la relation adresse MAC-adresse IP que les systèmes d'extrémité créent.

Les données, stockées dans la mémoire adressable par contenu* (CAM, de l'anglais *Content Addressable Memory*), déterminent où livrer les paquets entrants. Chaque fois qu'une partie du trafic arrive, le commutateur tente de déterminer sur quel port se trouve

---

* Comme son nom l'indique, ce type de mémoire peut être directement adressé par le paramètre dont vous essayez de déterminer la valeur, ce qui représente un gain de temps puisqu'il n'est plus nécessaire de rechercher ce paramètre. Un catalogue de bibliothèque est un exemple simple de CAM : vous n'avez pas besoin de passer en revue tous les livres de la bibliothèque pour en trouver un ; vous choisissez où regarder en fonction de ce que vous recherchez (un morceau d'information sur le "contenu").

le destinataire. Si cette information est disponible, le paquet est livré directement (et uniquement) à ce port en particulier et garde les informations hors de portée des autres, ce qui améliore les performances du réseau.

Les réseaux virtuels et la gestion du trafic

Certaines solutions plus avancées de commutateurs fournissent des fonctionnalités supplémentaires destinées à faciliter la gestion de vastes réseaux et à diminuer les délais et les dépenses nécessaires au déploiement. Ces caractéristiques semblent aussi contribuer à la sécurité du réseau et peuvent comprendre les éléments suivants.

VLAN (Virtual LAN)

Ce nom générique désigne un ensemble de méthodes utilisées pour diviser un groupe de ports sur un dispositif physique en un ensemble de réseaux logiques distincts, ce qui isole le trafic sur un groupe de ports et empêche tout type de trafic de se croiser au niveau du commutateur (ce système est le plus souvent implémenté en utilisant la norme IEEE 802.1Q, que nous examinerons en détail dans la prochaine section). Implémenter un réseau local virtuel revient à scinder un seul commutateur en plusieurs périphériques totalement indépendants, à la différence de la solution VLAN, qui est beaucoup plus souple et économique car il est possible de transformer le réseau et de réaffecter les ressources matérielles à volonté. Les administrateurs réseau du monde entier ont accueilli chaleureusement les VLAN car ils offrent un moyen simple mais puissant de construire un ensemble de réseaux distincts sur un seul périphérique ou, par exemple, de séparer les serveurs des postes de travail sans avoir besoin d'acheter un commutateur dédié pour chaque groupe.

Le trunk

Il s'agit d'une extension naturelle de la conception de base de VLAN. Les trunks utilisent le balisage des trames IEEE 802.1Q pour regrouper des VLAN multiples sur une seule liaison au lieu d'obliger l'utilisateur à utiliser des raccordements différents sur chaque périphérique, comme illustré à la Figure 7.1. Les paquets Ethernet provenant de l'ensemble ou de certains des VLAN sur le commutateur source sont balisés avec suffisamment d'informations pour déterminer leur VLAN d'origine à l'intérieur de l'en-tête de trame Ethernet, sont transférés à l'autre extrémité par tunneling sur une liaison traditionnelle, sont décodés puis envoyés au VLAN de destination approprié. Bien que cette option entraîne généralement une baisse des performances par rapport à l'utilisation d'un câble distinct pour chaque sous-réseau, elle est beaucoup plus pratique. Souvent, les systèmes trunk disposent également de DTP (*Dynamic Trunking Protocol*), un protocole trunk qui intègre des procédures d'autoconfiguration et permet aux périphériques de découvrir et d'échanger automatiquement les trames encapsulées avec d'autres

périphériques sur lesquels le trunk est activé sans nécessiter aucune action d'administration particulière.

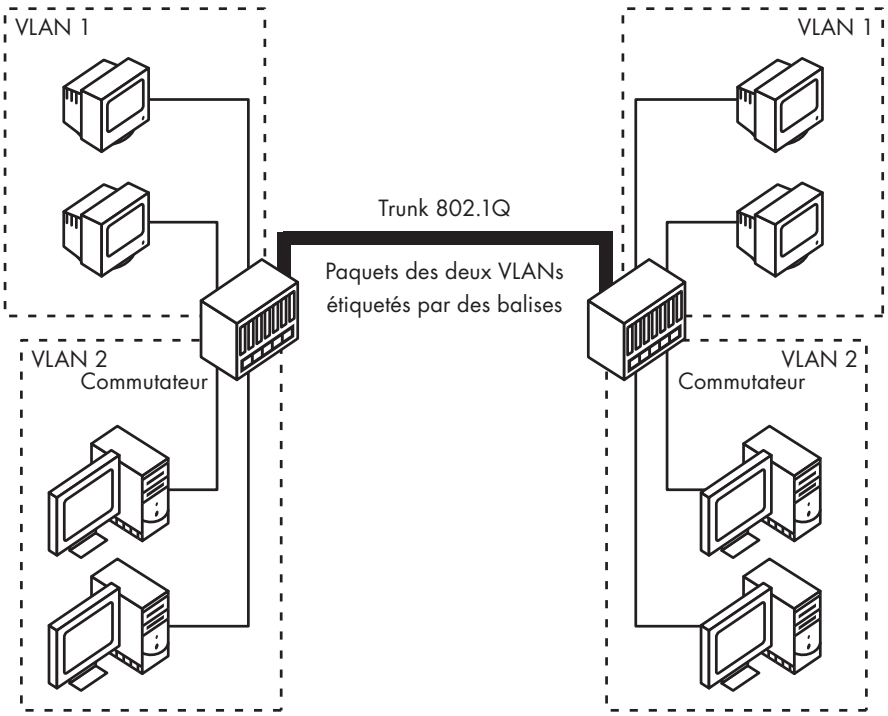


Figure 7.1

Le trunk VLAN en pratique. Les VLAN se propagent sur deux périphériques. Les périphériques sur toutes les occurrences de VLAN 1 et de VLAN 2 peuvent dialoguer entre eux, mais le dialogue croisé entre VLAN 1 et VLAN 2 n'est pas possible.

Le protocole STP

Le protocole STP (*spanning tree protocol*) permet de construire des structures de réseau redondantes dans lesquelles les commutateurs sont interconnectés dans plus d'un endroit, afin de maintenir une tolérance aux pannes. Traditionnellement, cette conception peut entraîner la mise en boucle infinie du trafic broadcast et de quelques autres paquets, tout en provoquant une chute significative des performances du réseau car les données reçues sur une interface et transmises à une autre rebondissent en effet vers l'expéditeur (voir le schéma de gauche de la Figure 7.2).

Lors de la conception d'un réseau, il est souvent difficile d'éviter les boucles broadcast accidentelles. Il est également parfois souhaitable de concevoir des architectures avec des boucles potentielles (dans lesquelles un commutateur se connecte à deux commutateurs ou plus), car ce type de conception a une meilleure tolérance aux pannes et permet qu'un seul périphérique ou un seul lien soit retiré sans scinder l'ensemble du réseau en deux parties totalement isolées.

Pour permettre la création de boucles et d'autres architectures complexes sans entraîner de sérieux problèmes de performances, le protocole STP implémente un mécanisme d'élection afin de choisir un nœud de commutation "racine". En fonction du résultat de cette élection, une hiérarchie arborescente du trafic est établie à partir de ce nœud et les liens qui pourraient provoquer une inversion de la propagation du trafic broadcast sont temporairement bloqués (voir le schéma de droite de la Figure 7.2). Vous pouvez rapidement changer cette hiérarchie simple qui s'établit de façon autonome lorsqu'un des nœuds connaît une panne et réactiver un lien précédemment jugé inutile.

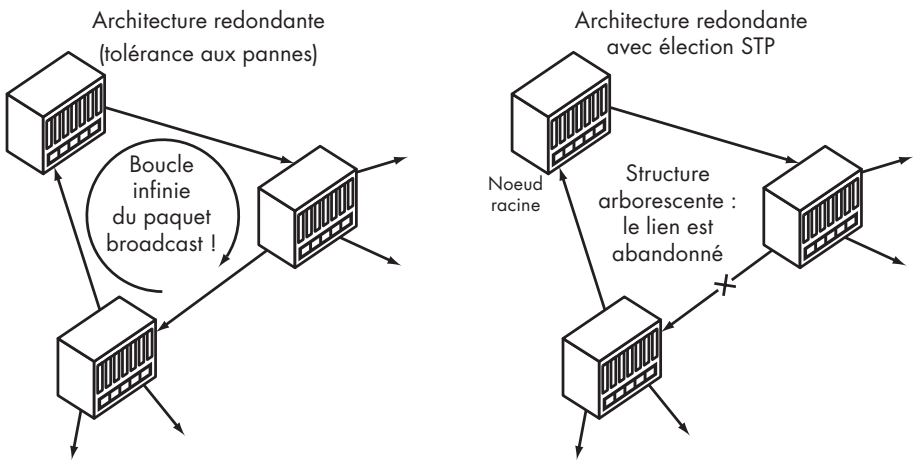


Figure 7.2

Le problème de tempête de diffusion et le schéma d'élection de STP. Le schéma de gauche illustre un réseau tolérant aux pannes sans STP, dans lequel certains paquets effectuent une boucle (presque) infinie entre les commutateurs. Le schéma de droite illustre le même réseau dans lequel un périphérique a été élu automatiquement comme nœud maître en utilisant STP et pour lequel la topologie logique a été ajustée pour éliminer les boucles. Lorsqu'une des liaisons est en panne, le réseau sera reconfiguré pour garantir la continuité des opérations.

Attaques sur l'architecture

Les mécanismes examinés jusqu'ici ont été conçus pour améliorer les performances de base tout en offrant des performances élevées, en se greffant sur une conception du réseau qui n'offre aucune fonction de sécurité[2]. Mais, si certaines attaques courantes comme le *MAC spoofing* (la possibilité pour toute personne de falsifier un message ARP et d'usurper l'adresse IP d'un périphérique en particulier) sont considérées par tous comme autant de failles des réseaux locaux, sont bien comprises et faciles à empêcher avec des commutateurs correctement configurés, d'autres graves défauts de conception ne sont pas si simples et ne peuvent donc pas être contrés aussi facilement. Il n'est pas toujours évident que les solutions courantes visant à améliorer la sécurité ne soient en fait d'aucune aide dans ce domaine.

Le CAM et l'interception du trafic

L'exemple du débordement du CAM est une des raisons les plus spectaculaires de ne pas considérer les commutateurs comme une fonction de sécurité. Le CAM qui stocke les associations entre les adresses MAC et les ports a une taille fixe et limitée généralement construite de manière non discriminatoire. Chaque fois qu'un système ne peut pas être localisé dans le CAM, le commutateur ne dispose que d'une seule façon pour livrer le paquet, il doit utiliser le mode de diffusion du concentrateur et donc envoyer le paquet à tous les systèmes, en espérant que le destinataire reconnaisse que ce trafic lui est adressé et que les autres systèmes soient assez gentils pour le négliger complètement. Ainsi, un attaquant attentif peut employer une tactique visant à générer un grand nombre de requêtes et de réponses ARP fausses ou d'autres paquets, en usurpant l'identité d'un grand nombre de périphériques distincts du réseau, juste pour remplir le commutateur du CAM. Une fois le CAM plein, l'attaque a effectivement dégradé la sécurité du réseau en désactivant le routage intelligent des trames sur le commutateur et le force à recourir à la diffusion de toutes les données. Cela, à son tour, permet à l'attaquant d'écouter toutes les communications, comme si le réseau n'était pas commuté du tout. L'attaquant peut faire tout cela sans usurper l'identité du destinataire ou affecter de manière visible le fonctionnement du réseau, si bien que la victime peut très bien rester totalement ignorante de ce problème. Il s'agit d'un problème de conception, non pas d'un défaut lié aux objectifs que doivent remplir les commutateurs, mais d'une mauvaise compréhension du fonctionnement de ces périphériques. Soyez rassuré, il est presque impossible de résoudre totalement ce problème dans un environnement typique. Certains commutateurs implémentent effectivement des limites de temps et des limites sur les ports pour empêcher de telles attaques, mais celles-ci ne sont jamais à 100 % efficaces.

Autres exemples d'attaques : DTP, STP, trunks

D'autres problèmes sont généralement plus faciles à empêcher et restent plus évidents (ils peuvent souvent être détectés par la victime), mais ils illustrent également les questions de sécurité au niveau d'Ethernet. Par exemple, une attaque sur le mécanisme DTP dont nous avons parlé est une possibilité intéressante. Le système de négociation automatique de DTP est souvent activé pour tous les ports d'un périphérique afin de faciliter l'installation. Le problème est qu'un attaquant habile peut donc prétendre être un commutateur sur lequel le trunk est activé et non un simple utilisateur final d'un poste de travail ou un modeste serveur. Une fois reconnu par le commutateur auquel il est connecté comme étant un périphérique ami, il commence à recevoir des trames balisées 802.1Q, y compris le trafic provenant d'autres réseaux locaux virtuels et desservis par le commutateur auquel il est connecté, ce qui le rend capable d'intercepter ou d'injecter du trafic malveillant sur des réseaux avec lesquels il n'est pas censé être en mesure de communiquer. Dans de nombreux réseaux où le même commutateur gère à la fois les réseaux protégés et "démilitarisés" ainsi que l'infrastructure commune du réseau LAN de l'entreprise, une telle attaque peut permettre d'obtenir des données très utiles en permettant aux membres de l'un des réseaux d'espionner ou d'interagir avec les membres de l'autre réseau.

Vous pouvez résoudre ce problème DTP sur certains périphériques en changeant leur configuration par défaut et en définissant clairement un ensemble de ports dédiés sur lesquels le trunk est activé sur le commutateur. Cependant, le problème ne s'arrête pas là. Notre autre ami, STP, peut être victime d'une attaque similaire, en permettant à un attaquant de se choisir comme commutateur "racine", et donc recevoir une partie du trafic réseau. Désactiver la découverte STP peut même se révéler encore plus difficile dans un environnement typique d'entreprise.

Un autre problème se pose lorsqu'un trunk quelconque a pour origine ou destination un VLAN non dédié (lorsque le port utilisé pour le trunk est placé dans un VLAN également utilisé par les postes de travail). En injectant des trames déjà balisées, il est possible d'injecter du trafic sur un trunk. On peut considérer cela comme un défaut de configuration ; pourtant, ce problème est souvent négligé car de nombreux ingénieurs estiment que la méthode d'implémentation des trunks est beaucoup plus avancée et magique qu'elle ne l'est en réalité.

Prévention des attaques

Ces problèmes sont souvent difficiles à résoudre, en particulier si toutes les phases de développement et d'expansion du réseau n'ont pas été strictement et étroitement surveillées. Bien que certains périphériques haut de gamme fournissent des fonctionnalités de sécurité étendues pour contrer les vecteurs d'attaque potentiels et atténuer ou

éliminer certains des risques, les réseaux Ethernet n'ont pas été conçus pour assurer la sécurité, et les dispositifs intelligents pour gérer ces réseaux sont rares. L'attaquant peut facilement rendre une partie ou la totalité de ces fonctionnalités inutile et dégrader le modèle de sécurité du réseau de la manière la moins souhaitable.

Même s'il existe des méthodes et des pratiques strictes à suivre pour sécuriser un réseau Ethernet, la complexité de ce processus, le coût financier supplémentaire et l'impact sur les performances sont souvent tels, sans compter le nombre de vecteurs à traiter, qu'il est évident que cette technologie n'a pas été conçue en pensant à la mise en pratique d'un quelconque niveau de sécurité.

Matière à réflexion

Lors du développement d'Ethernet, il semblait raisonnable de ne pas prendre en considération la sécurité dans les décisions de conception et de laisser la charge de sécuriser le réseau à une architecture de plus haut niveau comme le cryptage, entre autres. Au fil du temps, toutefois, cette décision initiale a contribué au coût global de maintenance des réseaux Ethernet et les a rendus difficiles à protéger des attaques sans devoir sacrifier de fonctionnalité d'une façon ou d'une autre.

Le problème ne se limite pas seulement à Ethernet. De nombreux réseaux conçus pour être dignes de confiance en se reposant sur des critères d'accès physiques ou matériels, y compris la plupart des systèmes téléphoniques du monde, par exemple, sont par nature exposés à des menaces internes sans qu'il existe de moyens ou presque de contenir efficacement l'exposition et de contrôler les dommages collatéraux qui apparaissent lorsqu'un seul système est compromis au sein du réseau. À mesure que la taille du réseau et le nombre des échanges augmentent, la probabilité qu'un des systèmes soit exploité par un utilisateur malveillant ou soit insuffisamment protégé des accès physiques ou distants devient de plus en plus forte. Bien qu'il soit normalement nécessaire d'accéder au *backbone* plutôt qu'à la station de travail d'un utilisateur final pour compromettre le système – ce qui rend la situation quelque peu différente d'Ethernet –, les systèmes actuels VoIP (*Voice over IP*) permettent souvent très facilement d'utiliser les techniques d'usurpation et d'autres subterfuges car ils font trop confiance à l'utilisateur.

8

L’enfer, c’est les autres

Que peut-il se passer d’autre dans le périmètre de "notre" réseau local ?
Beaucoup de choses !

Les réseaux locaux, qu’il s’agisse d’un réseau Token Ring ou plus couramment d’un réseau Ethernet, ont été conçus en supposant qu’il n’était pas nécessaire d’assurer la sécurité au niveau (ou sur la couche) de la technologie utilisée pour transmettre les données elles-mêmes. Au temps des premiers ordinateurs, les utilisateurs qui partageaient un réseau étaient tous censés être gentils.

Même si cette seule raison suffit pour comprendre que les concepteurs d’Ethernet n’aient pas vu le besoin d’intégrer des fonctionnalités de sécurité complètes dans leur conception, ils restent coupables d’avoir fait preuve d’un optimisme injustifié et de ne pas avoir prévu l’inévitable. Ethernet ne laissait tout simplement pas la place d’implémenter facilement des mécanismes de vérification de l’identité de l’expéditeur, de l’intégrité et de la confidentialité des transmissions ni sur les couches supérieures du modèle OSI, ni sur les périphériques, ni sur les applications. Les systèmes de protocoles et de communication suivants tentèrent d’implémenter un respect partiel de la confidentialité et de garantir l’identité des communications, mais ne parvinrent qu’à montrer un peu plus l’impossibilité de mettre en œuvre une sécurité suffisante sans revenir en arrière et retravailler la couche de liaison. La seule possibilité qui restait consistait à établir des astuces cryptographiques complexes très coûteuses en termes de calcul par-dessus le

système existant. Et cette complexité contribue à un certain nombre de problèmes de sécurité que l'on découvre, année après année.

Cette tendance regrettable a effectivement créé un ensemble de mécanismes de mise en réseau qui, même s'ils fonctionnent bien et sont abordables, ne sont pas appropriés pour gérer des données même moyennement sensibles lorsqu'une partie hostile est présente (presque toutes les données relatives à l'utilisateur transmises sur un réseau local sont sensibles). Les solutions qui essaient de résoudre ces problèmes, comme les applications de réseau privé virtuel (VPN, de l'anglais *Virtual Private Network*), l'encapsulation cryptée pour quelques-uns des protocoles Web les plus populaires ou encore les commutateurs avancés, sont habituellement plus coûteuses et sophistiquées qu'elles n'auraient pu l'être si la sécurité avait été au centre des préoccupations lors de la conception initiale du système de communication Ethernet.

Avant d'en arriver là, nous avons vécu dans le déni partiel pendant un bon moment. Lorsque la sécurité devint une vraie préoccupation (avec l'expansion d'Internet et la prolifération soudaine des systèmes compromis), les premières défenses à apparaître se concentraient sur le monde extérieur, tout en ignorant les menaces qui pouvaient venir de l'intérieur du réseau, auquel on faisait "confiance". Mais plusieurs entreprises et organismes institutionnels ont rapidement appris quelques leçons à leurs dépens. Avec le temps, il devint évident que les défenses externes comme les pare-feu et les systèmes de détection des intrusions ne suffisaient pas, même s'ils étaient correctement configurés dans l'ensemble de l'entreprise. La couche réseau est encore vulnérable, ce qui permet à un intrus de compromettre les échanges de données sans exploiter les failles de sécurité d'un seul système de l'entreprise.

Même si le réseau pourrait être sécurisé en déployant des mécanismes appropriés de chiffrement cryptographique et de vérification d'identité sur toutes les interfaces, cela est souvent peu pratique, voire impossible à réaliser, surtout sans avoir d'influence sur les performances ou la fiabilité du réseau et sans entraîner de coûts importants (sans parler des questions de compatibilité entre les différents systèmes d'exploitation et programmes). Par ailleurs, comme je l'ai mentionné, la cryptographie n'est pas toujours la réponse : non seulement il est beaucoup plus facile de réussir une attaque lorsque les données peuvent être vues et interceptées (attaques par relecture ou fondées sur le temps, par exemple), mais certains types d'informations, comme le défaut de remplissage des trames Ethernet que nous avons abordé plus tôt, peuvent aussi déjouer tous les efforts visant à protéger l'utilisateur.

Dans la deuxième partie de ce livre, nous abordons certaines des menaces inhérentes aux réseaux locaux qui révèlent des informations sans qu'aucune attaque classique ne se produise jamais. Tous ces problèmes subsisteront tant que les réseaux conserveront cette conception ancienne assez mal adaptée à la mise en réseau de nos jours.

Nous sommes maintenant prêts à aller de l'avant mais, avant de nous plonger dans le monde sauvage et fascinant situé au-delà du périmètre local, examinons d'autres cas intéressants (et plus précis) d'exposition.

Les indicateurs logiques et leur utilisation inhabituelle

Cet exemple a trait à l'abus des indicateurs logiques, autrement dit de compteurs, flags et autres gadgets qui n'ont pas de représentation physique mais sont plutôt conservés par un ordinateur, mis à disposition dans un logiciel et couramment implémentés dans les réseaux locaux. Les indicateurs logiques sont des éléments utiles qui, une fois de plus, partent du principe que le réseau local est digne de confiance.

Le protocole simple de gestion de réseau SNMP[1] (*Simple Network Management Protocol*) est la méthode la plus courante pour surveiller et parfois administrer les périphériques réseau. Le protocole SNMP est souvent implémenté sur des systèmes d'extrémité (serveurs et postes de travail) ainsi que des périphériques réseau, comme les commutateurs, les routeurs et les imprimantes.

SNMP fournit un moyen de lire (ou de modifier) une représentation abstraite de nombreux systèmes et applications internes, des paramètres de fonctionnement et de configuration ainsi que des statistiques. En utilisant SNMP, vous pouvez interroger une imprimante réseau sur le nombre de cartes réseau dont elle dispose ou son temps d'exploitation puis utiliser exactement la même méthode pour effectuer une requête auprès d'un ordinateur central sur la même question, même si ces informations doivent être obtenues d'une façon totalement différente à l'intérieur de chaque système. Ainsi, SNMP permet de surveiller et de gérer facilement des environnements hétérogènes sans devoir implémenter une multitude de protocoles d'accès natif et de vérifications de procédures.

Naturellement, SNMP lui-même pose beaucoup de problèmes de sécurité en termes d'implémentation et de déploiement, mais ce n'est pas la question ici. Même correctement implémentée, cette fonctionnalité peut entraîner des failles de sécurité et divulguer certaines informations, en fournissant par exemple un accès en lecture seule aux statistiques apparemment sans pertinence d'une interface réseau (ce trou est éliminé si le protocole est soigneusement limité, mais cela est souvent impossible sur certains types d'équipements de réseau). Un attaquant attentif peut observer soigneusement les compteurs de trames ou de paquets sur un système qui exécute SNMP et utiliser ces informations pour établir un profiling suffisant et effectuer des attaques fondées sur le temps et ainsi récupérer des informations sur une session interactive ou d'autres caractéristiques intéressantes, d'une manière analogue à l'approche dont nous avons parlé au Chapitre 1.

Oups ! Mais, à vrai dire, cela peut-il causer vraiment beaucoup de problèmes ?

Montrez-moi ce que vous tapez et je vous dirai qui vous êtes

Bien que j'aie déjà mentionné ce type de problèmes à plusieurs reprises et qu'ils puissent sembler abstraits, leurs conséquences sont bien réelles, même sans tenir compte de la reconstruction de la frappe, sur laquelle je me suis concentré au Chapitre 1. Par exemple, un groupe de chercheurs allemands de l'université Regensburg (Institut für Bankinnovation) a créé un produit commercial, PSYLock, qui fournit une reconnaissance biométrique de l'utilisateur à partir de ce qu'il tape au clavier[2] : Avec PSYLock, ils ont été en mesure d'identifier sans équivoque (et donc éventuellement de suivre à la trace) des utilisateurs en examinant la façon dont ils utilisent le clavier.

PSYLock repose principalement sur la mesure du temps qui s'écoule entre la frappe de deux touches, une astuce dont j'ai parlé plus tôt. Étant donné la possibilité d'observer les compteurs de paquets d'une machine en particulier et de calculer quand une touche est enfoncée par l'utilisateur dans une session interactive, on peut identifier une personne quel que soit le terminal qu'elle utilise. En appliquant ce concept à la couche réseau, on peut penser à certaines applications intéressantes, aussi bien dans un but malveillant que pour la surveillance. Si l'attaquant sait qu'il existe une session interactive de certains protocoles d'accès distants avec une station dont il peut surveiller les statistiques SNMP des ports, il peut déterminer quand les touches sont enfoncées en sondant plusieurs fois le compteur et donc savoir ce qui est saisi ou qui est en train de saisir.

Une variante plus légère de cette attaque est également possible. Et elle ne demande pas de recourir à une modélisation avancée, contrairement à ce que nous avons vu auparavant. Dans leur message posté sur Bugtraq et intitulé "Passive Analysis of SSH (Secure Shell) traffic"3 (analyse passive du trafic SSH), Solar Designer et Dug Song indiquent une autre attaque possible en utilisant cette fois le protocole SSH, une méthode courante de connexion à un système distant. Bien que SSH soit chiffré, il est possible dans les versions antérieures à leur recherche de mesurer la longueur d'un mot de passe en analysant soigneusement la taille d'un paquet observé au cours de la connexion (le mot de passe est envoyé dans un seul bloc de données entré une fois par l'utilisateur).

Cette technique pourrait également être appliquée avec succès à d'autres protocoles cryptographiques qui cachent la longueur d'un mot de passe en le bourrant avant de l'envoyer. Et il n'est pas surprenant que cette attaque puisse être menée simplement en observant le compteur d'octets SNMP plutôt qu'en surveillant directement le trafic.

Les bits inattendus : des données personnelles disséminées partout

Une autre raison explique que nous ne devrions pas être ravis à l'idée qu'une partie adverse espionne notre réseau (que nous croyions ou non que les données qu'ils peuvent voir sont sensibles). En effet, beaucoup de logiciels violent le principe de moindre surprise. Cette règle fondamentale dans la conception de logiciels dit qu'un programme doit réagir aux actions de l'utilisateur de façon à le surprendre le moins possible, autrement dit d'une manière cohérente, intuitive et prévisible (ou du moins attendue). En fait, beaucoup de programmes envoient bien plus d'informations précieuses que l'on pourrait penser et mettent ainsi souvent les utilisateurs dans une situation qu'ils n'avaient pas envisagée. Comme toujours, les programmes de Microsoft Windows se distinguent en diffusant beaucoup d'informations de façon intentionnelle mais souvent négligée et peu évidente. Mais le géant du logiciel convivial n'est pas le seul.

Bien que peu d'utilisateurs le savent, lorsque Windows travaille dans un domaine et qu'il est configuré pour utiliser des profils itinérants pour permettre à l'utilisateur de se connecter et d'accéder à ses données personnelles à partir d'un autre poste de travail, de vastes portions du registre de l'utilisateur sont envoyées au contrôleur du domaine à chaque fois qu'il se connecte ou se déconnecte. Bien que les informations contenues dans le profil puissent sembler tout à fait inutiles à première vue, elles contiennent différents paramètres personnels et renseignements historiques qui peuvent être très intéressants, notamment les dernières commandes exécutées, les dernières pages Web visitées et les derniers documents ouverts.

De même, ce qui est peut-être encore plus étonnant, si le répertoire de l'utilisateur dans le domaine réside sur un lecteur réseau, Windows cherche toutes les commandes entrées par l'utilisateur avec la commande Exécuter sur le serveur distant, puis en local. Ainsi, les informations sur toutes les commandes exécutées par l'utilisateur sont divulguées *via* le protocole SMB (*Server Message Block*) à un observateur attentif.

Ces quelques exemples parmi beaucoup d'autres montrent à l'évidence que la quasi-totalité des données sur le réseau doit être considérée comme sensible. En tant que tels, les réseaux locaux en général ne sont pas particulièrement bien adaptés pour transmettre des données récurrentes, sauf pour certaines configurations spécifiques, limitées ou qui disposent de moyens de protection supplémentaires. Et il n'existe aucun moyen sûr de protéger ces informations sans recourir à l'artillerie lourde, comme les tunnels IP chiffrés ou les solutions logicielles similaires, à moins de revoir tous les aspects de conception de la mise en réseau à partir de zéro.

Les failles du wi-fi

Il serait injuste de clore ce chapitre en ignorant le problème du remplaçant sans fil d'Ethernet : le wi-fi.

Les réseaux sans fil fondés sur le protocole IEEE 802.11 se généralisent dans le monde de l'entreprise et chez les particuliers. Malheureusement, avant même d'être aussi répandu et même s'il a été conçu dans le but d'apporter un niveau de sécurité plus important que les liaisons câblées, le wi-fi se révèle assez difficile à déployer correctement, sans doute parce qu'il a tenté de suivre les traces de son frère aîné d'un peu trop près.

Dans son principe de fonctionnement, la norme 802.11 n'est pas très différente de celle du réseau Ethernet. Elle utilise un système de contrôle d'accès traditionnel ("un membre parle, les autres écoutent"), mais le signal est transmis non par une paire torsadée de fils mais par une fréquence radio donnée. Ce qui nous amène au premier problème de la norme 802.11.

En mai 2004, le centre de recherche de l'université de technologie du Queensland (Queensland University of Technology Information Security Research Centre, ISRC) a annoncé avoir découvert que toute transmission sur un réseau d'entreprise 802.11 peut être interrompue en quelques secondes en transmettant simplement un signal qui empêche les autres membres du réseau d'essayer d'émettre. Bien sûr, il en est de même pour Ethernet, sauf que l'attaquant doit alors d'abord pouvoir se connecter à une prise réseau, ce qui le rend beaucoup plus facile à suivre et rend le problème plus facile à résoudre. Il suffit de vérifier le commutateur puis de suivre le câble. Cette attaque n'est pas vraiment surprenante mais ne correspond pas non plus à ce qu'attendaient les entreprises de cette technologie en l'adoptant.

Mais ce n'est pas tout. Lorsque la norme 802.11 tente de déjouer les attaques au niveau de la ligne de transmission, elle échoue lamentablement. Le mécanisme WEP (*Wired Equivalent Privacy*) a été conçu pour que les réseaux wi-fi fournissent un niveau de protection contre les écoutes clandestines des sessions réseau, ce qui offre une sécurité à peu près comparable à celles des LAN traditionnels. Or un certain nombre de défauts de conception du WEP ont été découverts en 2001 par des chercheurs de l'université de Californie et de l'entreprise Zero Knowledge Systems, prouvant que ce mécanisme n'était pas adéquat. Malheureusement, le wi-fi s'était déjà suffisamment répandu pour rendre difficile l'implémentation des modifications nécessaires[4].

Pour comble, l'utilisation du protocole WEP est optionnelle et désactivée sur la plupart des périphériques de réseau sans fil, les rendant ainsi prêts à accepter et à relayer tout le trafic qu'ils reçoivent. Bien que ce soit généralement acceptable pour les réseaux câblés,

pour lesquels une couche supplémentaire de sécurité est assurée sur le plan physique, les réseaux sans fil sont ouverts à tout intrus se trouvant à proximité.

Le *wardriving* (sillonner en voiture les villes à la recherche de points d'accès sans fil à l'aide d'un ordinateur portable compatible wi-fi) est devenu très populaire lorsqu'on découvrit que la majorité des réseaux sans fil des grandes entreprises, en particulier dans

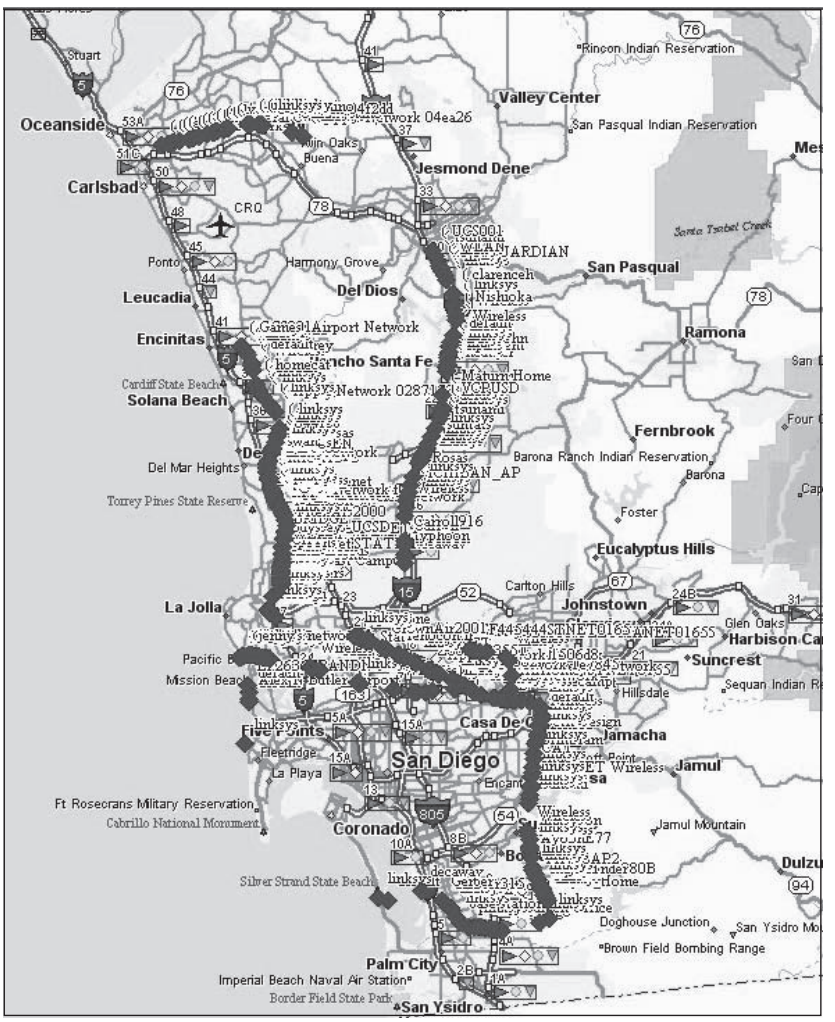


Figure 8.1

Les expéditions de warflying de Tracy Reed. Reproduites avec la permission de Tracy Reed, de Copilot Consulting (treed@copilotconsulting.com).

les grands centres commerciaux et les quartiers commerçants de chaque ville, étaient partiellement ou entièrement ouverts. Les abus sont souvent assez insignifiants et vont de l'utilisation gratuite du réseau à l'envoi de mailing de masse en passant par la réalisation d'attaques à distance par le biais du réseau de la victime, mais le risque que le réseau soit attaqué de l'intérieur par un attaquant doué est réel.

Quelle est la véritable ampleur du problème ? Il suffit de dire qu'à un moment le wardriving fut dépassé par le *warflying* (identique au wardriving mais en avion). En 2002, Tracy Reed, de Copilot Consulting, décida de voler autour de San Diego avec un scanner sans fil. À 1 500 pieds d'altitude, il réussit à trouver près de 400 points d'accès configurés par défaut et offrant probablement un accès libre au réseau Internet ou aux réseaux internes des entreprises pour toute personne se trouvant à proximité (voir Figures 8.1 et 8.2). Seuls 23 % des périphériques scannés étaient protégés par une clé WEP (qu'il est en général facile de casser de toute façon) ou de meilleurs mécanismes.

Allez comprendre.

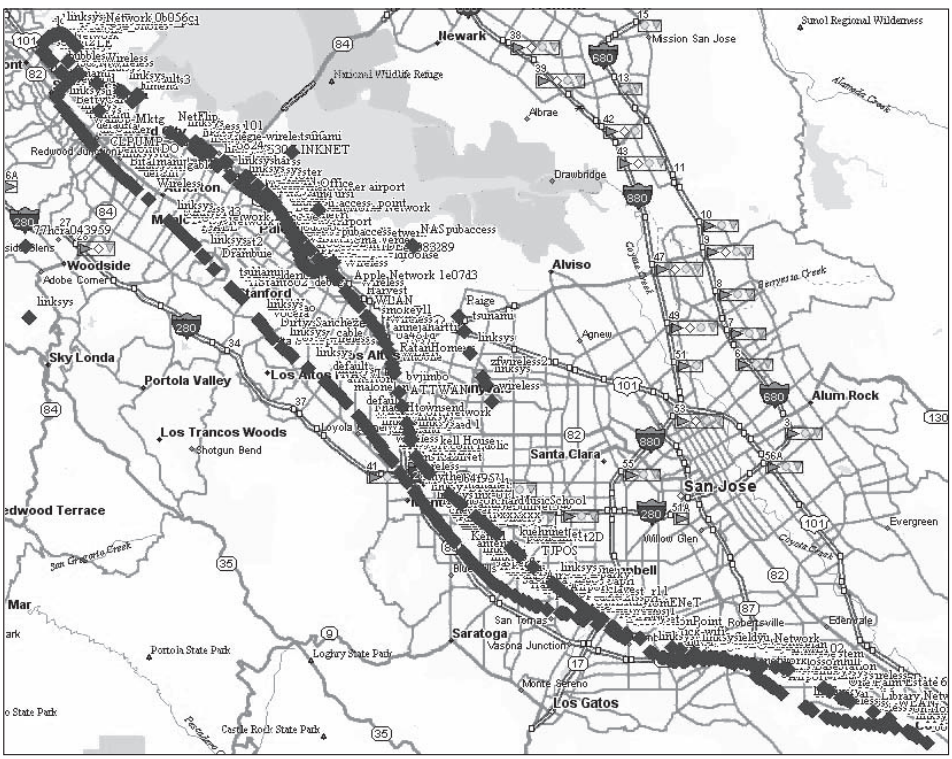


Figure 8.2

Warflying au-dessus de la Silicon Valley.

Partie III

Dans la jungle

Une fois sur Internet, les choses deviennent dangereuses.

9

Un accent étranger

Le fingerprinting passif : de légères différences de comportements peuvent aider les autres à savoir qui nous sommes.

Sur Internet, le réseau des réseaux, les informations envoyées à distance échappent au contrôle de l'expéditeur. Contrairement à un réseau Ethernet local qui est habituellement un endroit sûr pour les paquets tant qu'aucun intrus ne se manifeste, il n'est plus possible une fois que les données sont dans la nature d'évaluer ni de gérer efficacement les menaces qu'elles sont susceptibles de rencontrer, car une seule personne ne peut ni contrôler le chemin des données, ni connaître les intentions de toutes les parties impliquées dans la communication, ni encore moins déterminer la manière dont elles abordent la sécurité. Sur un réseau aussi complexe, le risque qu'une partie tierce soit malveillante n'est ni négligeable ni facile à évaluer. En fait, même la personne avec laquelle vous établissez une communication légitime peut avoir un but caché ou être tout simplement un peu curieuse.

Les tentatives d'acquisition de données non sollicitées sont également différentes lorsqu'elles se déroulent sur Internet pour quelques raisons supplémentaires. Surtout, elles n'ont pas besoin d'être ciblées et ne sont pas limitées à un segment de l'infrastructure physique. Comme l'attaquant n'a pas besoin de fournir de gros efforts pour les obtenir, elles représentent une manière potentiellement intéressante d'acquérir des données avant même de savoir précisément quels seront les avantages ou le profit

lucratif qu'il est possible d'en tirer. En outre, la ligne de démarcation entre les bons et les mauvais devient encore plus floue, car l'attaquant peut être votre meilleur ami. L'espionnage et la surveillance en général constituent un avantage si tentant en termes d'identification et de *profiling marketing* que beaucoup ont du mal à résister à la tentation d'en profiter. Le monde de la prestation de service n'est pas noir et blanc et, pour beaucoup de monde, avoir une moralité élastique n'est qu'une façon raisonnable de faire des affaires.

Cette partie de l'ouvrage se penche sur les menaces inhérentes à la conception ouverte d'Internet et sur la capacité des autres à obtenir plus d'informations sur vous que vous ne pourriez le penser (et beaucoup plus d'informations que nécessaire pour vous indiquer un site Web ou un jeu en réseau qui pourrait vous intéresser). Une fois sur Internet, l'ennemi n'est plus un fou isolé qui surveille les DEL du commutateur à travers la lentille d'un téléobjectif haut de gamme. Les cas d'exposition abordés ici permettent d'établir le profil des utilisateurs, de les suivre, de collecter des informations sur eux, rendent possibles l'espionnage industriel, la découverte d'un réseau et son analyse avant de réaliser une attaque. Ces menaces sont beaucoup plus réelles que les exemples décrits précédemment.

Vous devez comprendre ces menaces afin de maintenir un bon niveau de protection de la vie privée ou surveiller efficacement qui s'approche de vos systèmes, qu'il s'agisse de vos utilisateurs ou de parfaits inconnus. Il est également essentiel de les comprendre pour conserver toute sa raison dans un monde où la frontière entre souci de protéger sa vie privée et paranoïa clinique est assez mince.

Je vais commencer par examiner une série de protocoles réseau de base utilisés sur Internet et leurs incidences sur la confidentialité.

Le langage d'Internet

Le langage officiel d'Internet est appelé Internet Protocol, et sa version la plus courante est la version 4. Ce protocole, spécifié dans la RFC793[1], fournit un moyen d'implémenter une méthode standardisée de transmission des données sur de grandes distances et sur divers réseaux le plus simplement possible. Les paquets IP constituent la troisième couche du modèle OSI, dont nous avons déjà parlé. Ils sont constitués d'un en-tête, qui contient les informations nécessaires pour acheminer une partie des données à destination (le destinataire) et une charge contenant les informations de couche supérieure qui suivent immédiatement les données d'en-tête.

Les informations de routage fournies par l'expéditeur dans le paquet IP avant de l'envoyer se composent de l'adresse de l'expéditeur et du destinataire ainsi que d'un ensemble de paramètres qui simplifie le processus de transfert de données ou améliore

sa fiabilité et ses performances. Quand une machine sur le réseau local veut communiquer avec une partie distante qui n'est pas directement accessible sur le câble (du moins pas à la connaissance de l'hôte), elle transmet un paquet IP contenant l'adresse du destinataire final encapsulée dans une trame d'une couche inférieure à un ordinateur local qu'elle pense être une passerelle vers le réseau où se trouve l'adresse de l'expéditeur. La machine passerelle n'est rien de plus qu'un périphérique qui possède plusieurs interfaces connectées à plus d'un réseau et qui sert de point de connexion entre eux. La passerelle doit savoir comment router le paquet vers le monde extérieur, quoi faire de ce paquet et qui doit recevoir ensuite les données si davantage de parties sont impliquées dans le transfert avant que les données n'atteignent le destinataire.

Les systèmes impliqués dans le routage du trafic, depuis la passerelle locale au réseau de destination, lisent les informations fournies sur la couche IP pour choisir la façon de relayer les données, en se fondant sur ce qu'ils savent de la façon d'accéder à certains réseaux (dans ce contexte, un réseau est défini comme un ensemble d'adresses réseau situées à un endroit précis).

Routage naïf

Dans sa forme la plus simple, un routeur utilise une table de routage fixe pour faire la distinction entre un ensemble de réseaux locaux (vers lesquels il peut acheminer les données directement) et le monde extérieur, qui est inconnu. Ainsi, tout le trafic destiné à l'extérieur du réseau local doit être relayé à un routeur supérieur dans la hiérarchie qui a probablement une meilleure idée de l'endroit où envoyer les données.

La Figure 9.1 montre un exemple de structure de routage. L'expéditeur (à gauche) essaie d'envoyer un paquet à un système dont l'adresse appartient au réseau C, un réseau que l'expéditeur ne connaît pas. Pour faciliter la livraison, l'expéditeur envoie le trafic à la passerelle locale en espérant qu'elle sache où chercher le destinataire. Cependant, ce système, le routeur 1, ne peut atteindre que le propre réseau de l'expéditeur et un réseau A, un autre réseau qui n'a rien à voir avec C. Comme la cible ne se trouve pas sur leur réseau local, le routeur décide qu'il serait préférable d'envoyer le paquet à un routeur WAN supérieur dans la hiérarchie (routeur 2) qu'il est en mesure d'atteindre localement.

De plus, ce périphérique n'a aucune connexion immédiate avec le réseau C ; il peut uniquement atteindre directement les hôtes sur les réseaux B et D. Cependant, il sait que le routeur 3 dessert l'adresse de destination et donc qu'il doit sûrement savoir ce qu'il faut faire.

Par conséquent, le paquet lui est transmis et le routeur 3 peut désormais transmettre le trafic local au destinataire final, à la grande joie de tout le monde.

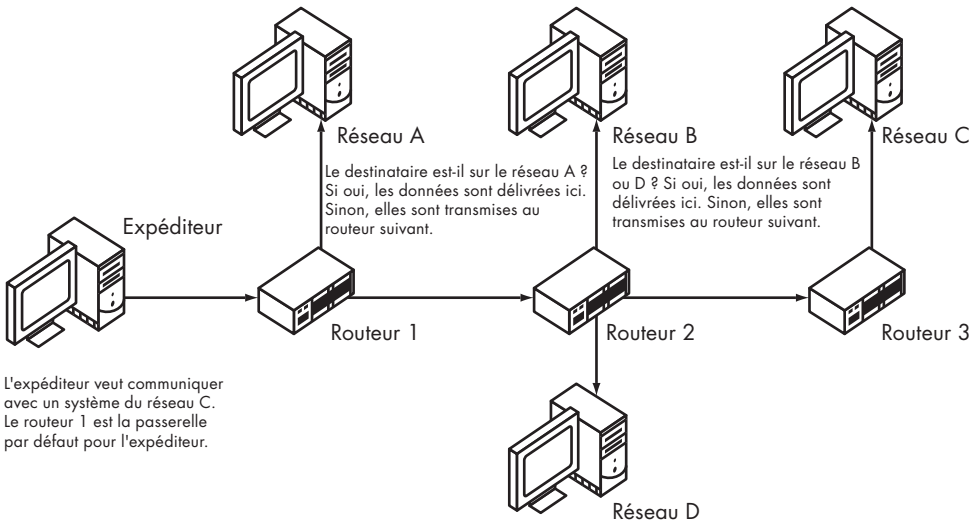

Réseau A
Réseau B
Réseau C
Le destinataire est-il sur le réseau A ? Si oui, les données sont délivrées ici. Sinon, elles sont transmises au routeur suivant.
Le destinataire est-il sur le réseau B ou D ? Si oui, les données sont délivrées ici. Sinon, elles sont transmises au routeur suivant.
Expéditeur
Routeur 1
Routeur 2
Routeur 3
L'expéditeur veut communiquer avec un système du réseau C. Le routeur 1 est la passerelle par défaut pour l'expéditeur.
Réseau D

Figure 9.1

Un schéma naïf de routage entre plusieurs réseaux étendus.

Le routage dans le monde réel

Dans la pratique, les réseaux sont souvent très redondants et ne possèdent pas une architecture strictement linéaire. Ils utilisent une structure arborescente complexe qui rend difficile la sélection du chemin le meilleur et le plus économique avec une configuration statique (sans parler du défi qu'il y aurait à rester à jour vu tous les changements d'infrastructure qui accompagnent le développement du réseau).

C'est pourquoi une stratégie de routage plus raisonnable est implémentée dès que le trafic atteint un *routeur backbone*. Dirigé par un opérateur de réseau, un routeur backbone est un périphérique WAN dédié qui relie de nombreux réseaux contrôlés par un fournisseur en particulier en une entité complexe appelée *système autonome*. Les routeurs backbone sont en général équipés d'interfaces avec d'autres routeurs importants et utilisent un algorithme avancé de découverte du chemin ainsi qu'un "annuaire" de noms et d'emplacement de réseaux d'une taille considérable. Ils sont contrôlés dynamiquement par un protocole BGP (*Boundary Gateway Protocol*) afin de trouver le meilleur moyen de router les données au système de destination, sans confier aveuglément la livraison du trafic à un système dans l'espoir qu'il soit en mesure de le relayer correctement.

L'espace d'adressage

Ce processus serait, bien entendu, tout à fait irréaliste si les réseaux de destination n'étaient qu'un ensemble d'adresses arbitrairement affectées à des périphériques à travers le monde. La liste de toutes les adresses d'un système autonome pourrait facilement représenter une taille énorme. Pour résoudre ce problème, des blocs continus de l'espace d'adressage sont affectés aux fournisseurs de services backbone, qui louent ensuite de petits blocs aux utilisateurs finaux ou à des prestataires de service moins importants. Le routage vers le fournisseur du réseau se déroule en recherchant l'adresse IP de destination dans les plages d'adresses affectées à cette entité, puis en effectuant une autre recherche au sein de ce réseau dans des tables de routage plus détaillées. Un système autonome peut donc être défini comme un ensemble d'adresses IPv4 (ou un ensemble de ces plages) utilisant un masque de sous-réseau.

L'adresse IPv4 utilisée pour identifier de manière unique un terminal du système dans toutes les communications IP a une structure assez simple. Elle est composée de 32 bits, divisés par commodité en 4 octets, soit un total de 4 294 967 296 adresses possibles. L'adresse est traditionnellement écrite avec des valeurs décimales comprises entre 0 et 255 et séparées par des points. Par exemple, 195.117.3.59 correspond à une valeur 32 bits de 3279225659.

Les blocs d'adresses IP continus constituent la base du routage des paquets. Elles sont définies au sommet de l'adressage IPv4 en définissant la part de l'adresse IP qui est fixe et constante pour tous les systèmes appartenant à un système autonome, ainsi que la partie de l'adresse qui sera fixée à différentes valeurs par le propriétaire d'un réseau afin de donner un identificateur unique aux ordinateurs.

Lors de la définition d'un réseau, certains bits plus significatifs d'une adresse IP (théoriquement, n'importe quel bit de 1 à 31 mais en fait les bits de 8 à 24) sont réservés comme adresse réseau. La partie fixe de cette adresse est partagée par toutes les adresses appartenant à (et vraisemblablement routées vers) ce réseau particulier. Le reste des bits moins significatifs peut être défini à volonté pour attribuer des adresses aux systèmes du réseau.

Historiquement (comme défini dans le RFC796[2]), la taille d'un réseau ou le nombre de bits significatifs verrouillés étaient une fonction de l'adresse qui pouvait être déterminée à partir de l'adresse du réseau lui-même. À partir des bits les plus importants de chaque adresse, les adresses furent regroupées pour constituer des réseaux de classe A (pour lesquels les 8 bits les plus significatifs sont fixes, ce qui permet au réseau de compter plus de 16 millions d'adresses possibles), des réseaux de classe B (pour lesquels 16 bits sont fixes, ce qui autorise plus de 65 000 hôtes) ou des réseaux de classe C (avec 24 bits fixes et 256 hôtes possibles). Par conséquent, si l'adresse IP de votre système commence

par le chiffre 1, il appartient à un réseau de classe A et tous les autres systèmes qui ont ce préfixe se trouvent proches du vôtre.

Bien que cela ait semblé suffisant à l'époque, l'espace d'adressage IPv4 a considérablement diminué dès les premiers jours d'Internet lorsque de nombreuses adresses réseau de classe A furent attribuées aux implémenteurs initiaux (l'armée américaine, Xerox, IBM et d'autres entités surpuissantes), qui ne semblaient pas disposés à les céder, même s'ils n'utilisaient qu'une fraction de l'espace dont ils disposaient pour des infrastructures publiques. En outre, une fois Internet devenu commercial et les adresses IP, une ressource que les utilisateurs devaient payer, les utilisateurs exigèrent un espace d'adressage qui corresponde mieux à leurs besoins. Certains ne voulaient que 4 adresses, tandis que d'autres voulaient un espace continu de 8 000 adresses. Les utilisateurs ont alors commencé à revendre ou à partitionner leur espace Internet.

Par conséquent, l'espace d'adressage actuel est partitionné de façon bizarre, de petites portions de l'espace d'adressage sont souvent exclues et reroutées depuis des blocs plus grands et continus, sans se soucier du partitionnement original. Chaque adresse de réseau est maintenant accompagnée d'une spécification de masque de réseau, car il n'est plus possible de dire à quel réseau un système appartient en se fondant seulement sur son IP. Les bits du masque de réseau sont définis à des positions qui devraient être fixes dans l'adresse du réseau et sont égales à zéro à des positions qui peuvent être manipulées librement au sein d'un réseau.

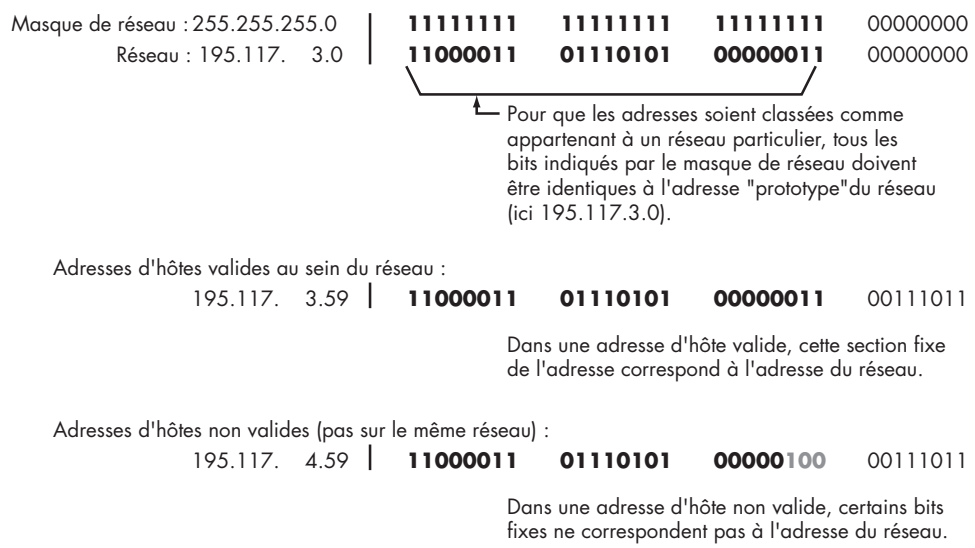

Masque de réseau : 255.255.255.0 | 11111111 11111111 11111111 00000000
Réseau : 195.117. 3.0 | 11000011 01110101 00000011 00000000
Pour que les adresses soient classées comme appartenant à un réseau particulier, tous les bits indiqués par le masque de réseau doivent être identiques à l'adresse "prototype"du réseau (ici 195.117.3.0).
Adresses d'hôtes valides au sein du réseau :
195.117. 3.59 | 11000011 01110101 00000011 00111011
Dans une adresse d'hôte valide, cette section fixe de l'adresse correspond à l'adresse du réseau.
Adresses d'hôtes non valides (pas sur le même réseau) :
195.117. 4.59 | 11000011 01110101 00000100 00111011
Dans une adresse d'hôte non valide, certains bits fixes ne correspondent pas à l'adresse du réseau.

Figure 9.2

Les règles d'adressage de réseau.

Comme le montre la Figure 9.2, en fixant 24 bits du réseau 195.117.3.0, 8 bits peuvent être changés. Cela permet de créer 256 adresses entre 195.117.3.0 et 195.117.3.255 sur ce réseau (bien que certaines implémentations obligent de réserver la première et la dernière adresse à des fins spéciales, ce qui ne laisse que 254 hôtes possibles). Avec une spécification d'un réseau d'adresses relativement aussi simple, il est facile de déterminer quelles adresses appartiennent à ce réseau et donc ce qui doit être livré à la passerelle de ce réseau (et ce qui ne doit pas l'être).

Bien que ce système d'adressage puisse sembler confus et inutilement compliqué, il fonctionne. Il permet d'associer des ensembles d'adresses à des systèmes spécifiques et de distinguer les systèmes informatiques les uns des autres avec un minimum d'efforts. Internet, dans toute sa complexité, réussit en général à trouver un système dans un très court laps de temps, sans trop de maintenance.

Des empreintes digitales sur l'enveloppe

Nous savons comment les données se rendent du point A au point B. Mais ce qui se passe en chemin est beaucoup plus intéressant que la façon dont le chemin est déterminé. Examinons donc de plus près ce qui est échangé entre les routeurs et nos systèmes d'extrémité. Bien qu'on puisse penser que la charge effective des données à l'intérieur des paquets envoyés sur Internet contient les informations les plus intéressantes (compte tenu de tous les courriels privés et des contenus bizarres échangés dans le monde à chaque seconde), il y a plus que cela.

Le format des paquets IP utilisés pour le routage des données et les informations de la quatrième couche utilisées pour encapsuler les données au niveau de l'application sont définis de façon assez stricte par les RFC et avec étonnamment peu d'ambiguïté. Toutefois, même avec une implémentation correcte de la pile TCP, les informations sous-jacentes peuvent avoir considérablement plus de valeur pour le destinataire que la charge des données qu'il reçoit. La divulgation à ce niveau est involontaire et inattendue mais, pour en savoir plus, nous devons examiner de plus près la conception des protocoles sous-jacents.

Internet Protocol

Commençons par le commencement. IP offre un mécanisme universel de livraison à longue distance fondé sur la troisième couche du modèle OSI. Il contient un ensemble de paramètres conçus pour être interprétés et éventuellement modifiés par des systèmes intermédiaires. L'en-tête est illustré à la Figure 9.3.

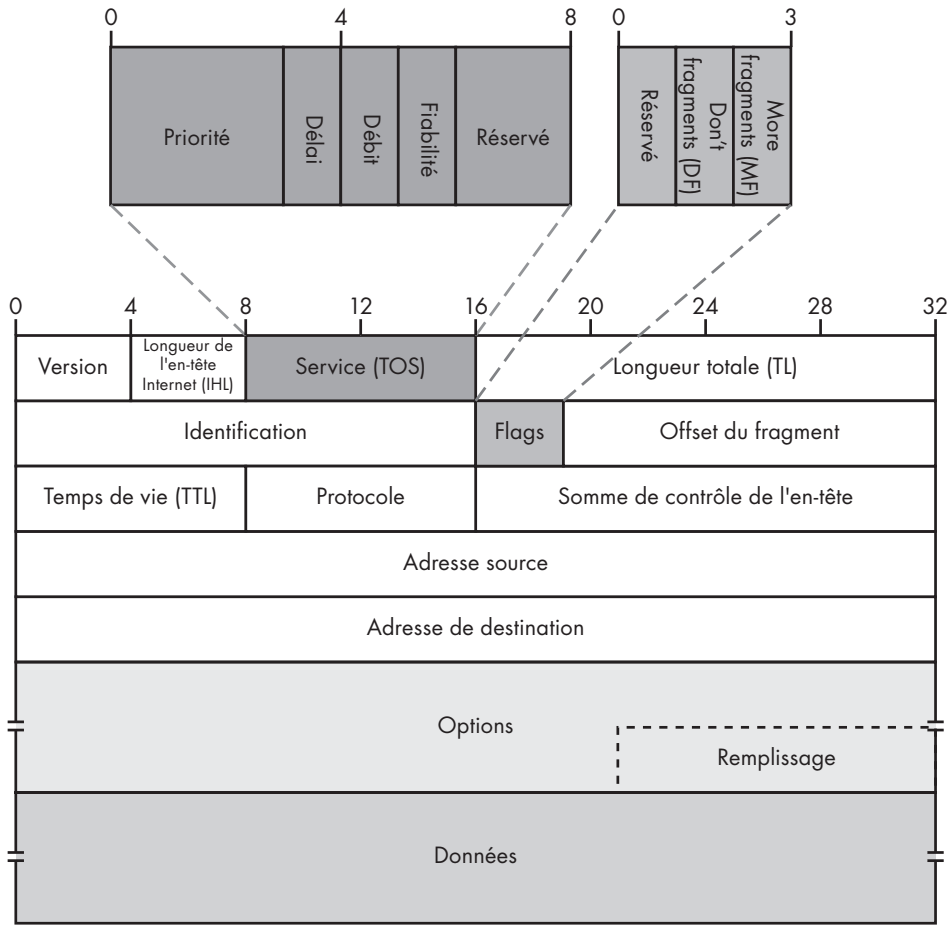


Figure 9.3

La structure d'en-tête IP.

Version du protocole

Il s'agit d'une valeur de 4 bits qui est fixée à 4 (0100) dans tous les paquets IPv4. IPv4 est le protocole standard (et dans de nombreux cas le seul pris en charge) de la troisième couche sur Internet. Les tentatives d'implémentations plus avancées, IPv6, n'ont pas été particulièrement fructueuses à ce jour (je pense que ce manque de succès est peut-être dû au fait que le nouveau format d'adresse IP est beaucoup plus difficile à mémoriser pour un administrateur système en général).

Le champ IHL

Il s'agit d'une valeur de 4 bits qui spécifie la longueur totale de l'en-tête IP. Elle représente la longueur en mots de 32 bits, exprimée en blocs de 4 octets (permettant d'exprimer des longueurs de 0 à 60 octets en utilisant les seize valeurs du champ). Ce paramètre indique à l'implémentation où cesser l'analyse de l'en-tête IP (qui peut avoir une longueur variable en raison des "options" supplémentaires qui peuvent être ajoutées à la fin de l'en-tête et immédiatement avant le contenu d'une couche supérieure). Il permet aussi de sauter une partie de l'en-tête IP sans avoir à regarder les options ou à les comprendre complètement pour passer directement aux données.

Comme les options IP ne sont généralement utilisées que pour des diagnostics (elles permettent par exemple le routage d'un paquet en particulier mais pas beaucoup plus), presque tous les paquets IP ont une longueur de 20 octets (ce qui signifie que ce champ est défini à 5), soit la longueur de la partie fixe de l'en-tête. Les valeurs inférieures à 20 sont évidemment erronées et le paquet est alors rejeté par une implémentation saine d'esprit (cependant, le bon sens n'est pas toujours la règle).

Le champ Service (8 bits)

L'importance de ce champ est généralement assez marginale. Il fournit une description de la priorité de routage fondée sur l'honneur dans laquelle l'émetteur est considéré comme agissant de bonne foi et autorisé à préciser si ce trafic revêt une importance particulière ou nécessite un traitement spécial. Cette valeur est parfois utilisée dans les installations locales, où ce niveau de confiance peut être exercé mais est souvent ignoré sur Internet.

Ce champ se compose de trois segments :

- Les trois premiers bits définissent la priorité.
- Les quatre bits suivants désignent la méthode de routage désirée (à l'aide de concepts abstraits tels que "haute fiabilité" ou "faible latence" en laissant au routeur le soin de les interpréter).
- La dernière partie, un seul bit, est réservée et doit être réglée sur 0.

La longueur totale (16 bits)

Ce champ de 2 octets représente la longueur totale de ce paquet IP, y compris sa charge. Bien que la plus haute valeur possible soit 65 535, la taille maximale d'un paquet est souvent limitée à une valeur beaucoup plus petite par les restrictions du protocole de plus bas niveau. Par exemple, Ethernet a un MTU (*maximum transmission unit*) de 1 500 octets.

Un système connecté à Ethernet n'enverra donc pas de paquets de taille supérieure. Des MTU supérieurs à environ 16 à 18 kilo-octets sont pratiquement inconnus. Les valeurs les plus courantes sont comprises entre 576 et 1 500 octets.

NOTE

Fait amusant : la taille limite d'un paquet IP, N octets (en fonction du paramètre MTU), impose également la limite minimale de la bande passante pour tout le trafic IP : 20 octets au moins seront ajoutés à l'en-tête par N-20 octets envoyés sur un niveau hiérarchique supérieur.

L'adresse IP source

Cette valeur 32-bit (une adresse IP dans le format abordé dans la section précédente) devrait représenter le système d'extrémité d'où provient la communication. Comme le paquet IP est établi par l'expéditeur et que personne n'est incité à vérifier l'exactitude de ce paramètre sur le réseau d'origine, on ne peut pas vraiment avoir confiance dans cette valeur seule. Elle fournit cependant une bonne indication sur l'identité de la personne à qui répondre (et, si l'on a une raison de faire confiance à cet indice, on peut l'utiliser pour répondre à l'expéditeur). L'acte de forger cette valeur intentionnellement est appelé communément *usurpation d'adresse IP* ou *IP spoofing*.

L'adresse IP de destination

Cette valeur 32-bit indique la destination finale du trafic. Comme tous les autres paramètres IP, il est choisi à la discrétion de l'expéditeur et utilisé par les systèmes intermédiaires pour diriger les paquets de façon appropriée.

L'identifiant du protocole de la quatrième couche

Il s'agit d'une valeur de 8 bits qui spécifie ce qui est transporté en tant que charge dans le paquet IP (les options TCP, UDP, ICMP et autres, dont nous parlerons plus en détail dans un moment).

Le TTL

Le TTL est un "compteur tueur" de 8 bits pour le trafic IP. Pour éviter les boucles infinies quand quelque chose tourne terriblement mal avec les tables de routage, le compteur est décrémenté d'une unité à chaque passage dans un système provisoire ou reste dans la file d'attente de transmission pendant une certaine durée. Lorsque le compteur atteint zéro, le paquet est rejeté, et l'expéditeur peut recevoir un message d'erreur au moyen d'un paquet

ICMP. La valeur TTL, comme toutes les autres, est choisie à la discrétion de l'expéditeur mais, en vertu de sa largeur de bit, ne peut pas être supérieure à 255.

Un effet secondaire intéressant du TTL compteur est qu'il peut être utilisé pour établir la cartographie du routage à un système distant : un message dont le TTL est de 1 expire au premier routeur qu'il rencontre sur son chemin (et l'expéditeur reçoit un message ICMP du routeur) ; un message dont le TTL est fixé à 2 expire au prochain saut, et ainsi de suite. En envoyant des paquets dont la valeur TTL augmente progressivement et en contrôlant l'origine des réponses ICMP indiquant que la durée de vie du paquet est dépassée, il est possible de cartographier l'ensemble des routeurs et des autres périphériques IP sur le chemin du destinataire. Cette technique baptisée *traceroute* est une méthode utilisée couramment pour diagnostiquer les problèmes de routage et effectuer des analyses avant une attaque.

Son utilité pour l'attaquant tient au fait qu'elle permet d'obtenir certains effets sans pour autant compromettre la victime : pour compromettre www.microsoft.com, vous pouvez cibler le routeur du réseau qui héberge ce serveur ou les routeurs de leurs fournisseurs d'accès, dans l'espoir d'intercepter tout son trafic et toutes les fausses réponses renvoyées. Cela aurait pour effet d'isoler le serveur puis, en usurpant son identité, de faire croire au monde extérieur que le site www.microsoft.com a changé. Naturellement, ce n'est qu'un exemple.

Les paramètres Flags et Offset

Ces valeurs 16 bits contrôlent un aspect du routage des paquets IP intéressant et comprenant peut-être plus de défauts. Ces paramètres sont utilisés chaque fois qu'un grand paquet doit être transmis par un système intermédiaire sur une liaison dont le MTU est inférieur à la taille du paquet. Dans un tel cas, la taille du paquet n'est pas "adaptée" au médium en l'état.

À titre d'exemple, un émetteur relié à Ethernet peut envoyer un paquet dont la taille est de 1 500 octets, ce qui est souvent le cas. Toutefois, si le premier routeur que le paquet rencontre établit un pont entre le LAN local et un modem DSL, un problème se pose : le MTU courant d'une liaison DSL (elle-même souvent une étrange combinaison d'encapsulations sur d'autres protocoles) est de 1 492 octets. Donc, un paquet de 1 500 octets n'est alors pas adapté.

Étant donné la grande variété de liaisons qui permettent le fonctionnement d'Internet de travail, il s'agit d'un problème grave. Pour le résoudre, on divise (*fragmente*) le paquet IP ou, plus précisément, sa charge en plusieurs paquets IP distincts et on ajoute des informations qui permettent aux destinataires de réassembler la charge avant de la transmettre aux couches supérieures. On obtient donc un nouvel ensemble de paquets adapté à cette liaison en particulier. Un offset défini sur chaque fragment indique la façon dont

chaque partie de la charge doit être insérée par le destinataire final pour reconstituer le paquet original.

Tous les fragments sauf le dernier disposent également du flag spécial MF (*more fragments*) dans leurs en-têtes. Lorsque le système de destination reçoit un paquet avec un flag MF ou un paquet dans lequel le champ Offset du fragment est défini mais sans flag MF (ce qui indique qu'il s'agit du dernier segment d'un paquet fragmenté), le système de destination sait qu'il doit allouer une zone de mémoire vide pour réassembler le paquet original et attendre tous les autres morceaux restants avant de poursuivre le traitement du paquet.

La Figure 9.4 montre le processus de fragmentation et de réassemblage au cours duquel un paquet surdimensionné est d'abord scindé en deux morceaux puis complètement réassemblé par le destinataire, en dépit du fait que les morceaux arrivent dans le désordre.

Bien que ce processus fonctionne, il est quelque peu inefficace. Il faut du temps pour que les systèmes fragmentent et réassemblent le trafic, et les derniers morceaux ne transportent souvent que peu de charge (uniquement les quelques octets qui n'ont pas leur place sur un autre type de liaison). Il vaut mieux bien sûr que l'émetteur soit en mesure de déterminer le plus faible MTU entre l'emplacement et la destination des paquets (également appelé path MTU, abrégé en PMTU) et de construire les paquets en conséquence. Malheureusement, le protocole IP n'offre pas de façon flexible et propre d'implémenter cela, mais cela n'a pas empêché des chercheurs de trouver un *hack*.

Selon ce hack, un système qui implémente la découverte PMTU place un flag DF (*don't fragment*, ne pas fragmenter) sur l'ensemble du trafic sortant. Si un routeur ne peut pas transmettre un paquet DF sans le fragmenter, il doit alors le rejeter et envoyer un message ICMP approprié qui indique : "Le paquet doit être fragmenté, mais paramétré DF." L'expéditeur, à la réception d'un tel message, peut ajuster ses attentes en conséquence, mettre en cache la découverte, et poursuivre avec des paquets dont la taille est plus appropriée.

NOTE

Cette pratique, définie dans RFC1191[3], suppose que le coût du renvoi du paquet abandonné est préférable à la dégradation constante des performances causée par la nécessité de la fragmentation. Cette technique, cependant, est également très controversée, car tous les périphériques n'envoient pas les notifications ICMP adéquates puisque cette fonctionnalité n'était pas exigée dans le passé. Ainsi, en activant le PMTUD (la découverte PMTU), un expéditeur peut être dans l'incapacité de communiquer avec certains sites ou les transferts de fichiers, être bloqués, ce qui est extrêmement difficile à diagnostiquer.

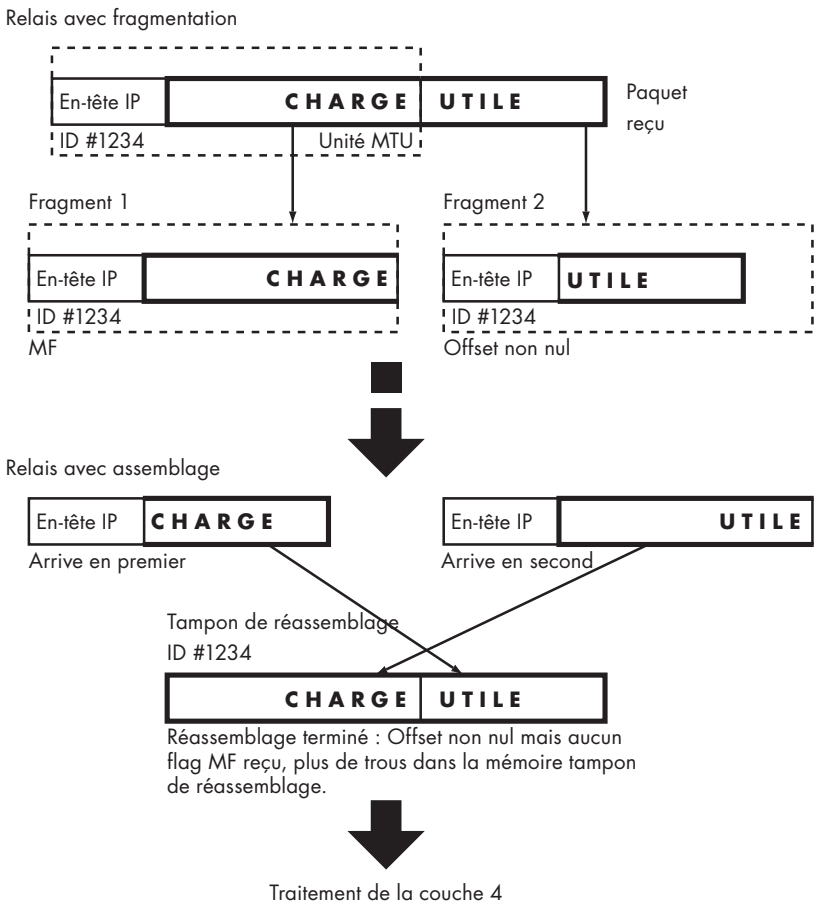


Figure 9.4

Le processus de fragmentation et de réassemblage des paquets.

Numéro d'identification

Le numéro d'identification (ID) est une valeur 16 bits qui différencie les paquets IP lors de la fragmentation. Sans cet IP ID, si deux paquets sont fragmentés simultanément, les fragments des deux paquets seraient altérés, interchangés ou du moins sérieusement endommagés lors du réassemblage.

Les IP ID identifient de façon unique une mémoire tampon de réassemblage différente pour chaque paquet. Pour cela, sa valeur est souvent simplement incrémentée à chaque

paquet envoyé ; le premier paquet envoyé par un système a un IP ID de 0, le deuxième, un IP ID de 1, et ainsi de suite.

NOTE

Sur les systèmes où le PMTUD est activé, les identifiants IP uniques ne sont pas nécessaires puisque, en théorie, la fragmentation ne se produit pas et que la valeur est souvent fixée à 0 (même si cela n'est sans doute pas particulièrement une bonne idée, car certains périphériques assez courants ont tendance à ignorer le flag DF).

Somme de contrôle

La somme de contrôle est un nombre de 16 bits qui fournit une méthode de détection simple des erreurs. La somme de contrôle doit être recalculée sur chaque saut (car des paramètres comme le TTL changent) et est donc conçue pour utiliser un algorithme rapide, ce qui n'est pas particulièrement fiable. Bien que de nos jours le terme "somme de contrôle" n'ait de somme que le nom (en utilisant des algorithmes comme CRC32 ou des fonctions de raccourci sécurisées par chiffrement), la somme de contrôle IP est en fait une somme, ou une variante de celle-ci, couplée à des négations bit à bit* injectées pour que la somme de contrôle ait moins de chances de rester correcte en cas d'erreur de transmission.

Au-delà du protocole IP

Une conséquence des nombreuses décisions de conception prises lors de l'élaboration d'IPv4 est l'absence d'une garantie de fiabilité raisonnable, même si le réseau lui-même se comporte de manière fiable. Bien que les numéros IP ID soient destinés à minimiser les risques de collisions lors du réassemblage, leur taille relativement faible de 16 bits (ce qui autorise 65 536 valeurs possibles) entraîne l'apparition de problèmes lorsque deux paquets ayant des IP ID identiques sont réassemblés simultanément. En outre, les sommes de contrôle des en-têtes IP ne suffisent tout simplement pas à fournir une protection de l'intégrité fiable ; même si cela est peu probable, un changement aléatoire dans un paquet pourrait toujours donner une somme de contrôle identique. De plus, si le réseau connaît une panne, il n'existe aucun moyen de savoir quelles données ont disparu, même si l'échec est simplement dû à une brève surcharge d'un seul composant du réseau.

* Techniquement parlant, bien que cela n'ait aucune importance particulière pour le sujet qui nous concerne, la somme de contrôle est constituée en calculant le complément à 1 sur 16 bits de la somme des compléments à 1 des octets de l'en-tête et des données pris deux par deux (mots de 16 bits).

Enfin, IP ne fournit aucun moyen de vérifier l'expéditeur d'un message et estime simplement que le véritable expéditeur est celui figurant dans l'en-tête IP. Les fonctionnalités de fiabilité et d'intégrité sont laissées aux protocoles de plus haut niveau au besoin (et, le plus souvent, cela est nécessaire). Par conséquent, des protocoles de plus haut niveau au sommet de l'IP sont nécessaires.

TCP et, dans une moindre mesure, UDP fournissent non seulement l'indispensable protection du trafic mais permettent également à l'utilisateur de définir le destinataire (ou l'expéditeur) autrement qu'en pointant simplement vers un système.

Tandis que l'en-tête IP contient uniquement assez d'informations pour router le trafic entre deux systèmes, mais pas assez pour décider à quelle application ces informations doivent être livrées, UDP et TCP vont un peu plus loin : ces protocoles agissent dans le terminal et indiquent au destinataire vers quelles applications diriger les données entrantes.

Le protocole UDP

Comme le définit le document RFC768[4], UDP (*User Datagramm Protocol*, ou protocole de datagramme utilisateur) fournit un minimum de fonctionnalités supplémentaires au protocole IP. UDP ajoute un mécanisme pour la livraison locale des données, mais son manque de fiabilité reste très proche du niveau de la couche sous-jacente (ainsi que de sa surcharge faible). L'utilisation d'UDP pour les communications peut être assimilée à un service téléphonique dans lequel les mots sont parfois intervertis ou les phrases, tronquées et pour lesquelles il n'existe pas d'identifiant fiable de l'appelant. Mais le coût d'un appel est faible et vos appels sont traités rapidement.

L'en-tête UDP (voir Figure 9.5) possède un minimum de caractéristiques et est relativement simple. Il introduit un petit ensemble de paramètres qui peuvent être interprétés par le système de destination et utilisés pour router un paquet vers une application spécifique ou pour vérifier que la charge des paquets n'ait pas été altérée en cours de route.

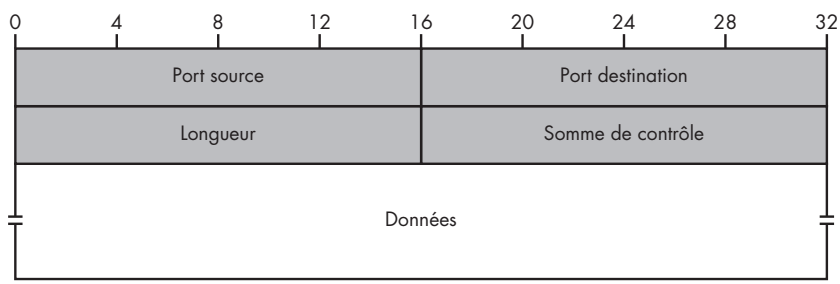


Figure 9.5

La structure de l'en-tête UDP.

UDP s'utilise pour les requêtes simples, dans des situations pour lesquelles il n'est pas nécessaire de conserver l'état des informations et quand les performances et le coût général sont plus importants que la fiabilité. UDP est par exemple couramment utilisé pour la résolution du nom des DNS (systèmes de noms de domaines), les protocoles réseau d'amorçage et d'autoconfiguration simples (BOOTP), les technologies de lecture en continu de flux audio ou vidéo, les systèmes de partage de fichiers en réseau, et ainsi de suite.

Introduction à l'adressage des ports

UDP introduit la notion de port source et de port de destination en plus des adresses sources et de destination, un concept qu'il partage avec TCP (protocole plus avancé de la quatrième couche que je vais aborder ensuite). Un port est un certain nombre de 16 bits qui est choisi par une application d'un terminal prêt à envoyer ou à recevoir des données ou qui lui est assigné par le système d'exploitation (appelé *port éphémère*).

Un port est un moyen de router les données d'une application ou d'un service spécifique sur un système multitâche afin que des communications simultanées puissent se produire entre les programmes. Par exemple, une application serveur peut décider d'écouter le port 53 pour les requêtes entrantes, tandis qu'un système de journalisation des installations peut écouter le trafic adressé au port 514. Les ports permettent aux clients de communiquer à ces processus en même temps. De plus, lorsque l'implémentation prend en charge une séparation nette entre les paires de ports source et de destination, il est possible que deux clients utilisent des ports source éphémères différents pour communiquer avec le même service (le port 514, par exemple) en même temps, sans que cela n'entraîne de problèmes pour savoir quelle application client doit obtenir quelle réponse du service distant.

Pour que le système de destination fasse la différence entre les communications adressées à une application particulière et les livre comme prévu, l'expéditeur doit indiquer le numéro du port de destination dans tout le trafic qu'il émet. L'expéditeur définit un port source différent pour chaque application client afin que la réponse du serveur soit ensuite livrée au bon composant.

Dans ce schéma d'adressage du port, un quadruplet de valeurs (hôte source, port source, hôte de destination, port de destination) est utilisé pour assurer une bonne séparation du trafic et une bonne gestion des sessions pour les connexions simultanées qui sont émises ou reçues sur un système spécifique. Cette conception signifie que près de 65 535* clients peuvent se connecter au monde extérieur à partir d'une seule adresse IP et que

* Ce nombre s'élève en fait à 65 536; mais le port numéro 0 ne devrait pas être utilisé. Le système d'exploitation et ses applications peuvent le permettre, évidemment, et donc être en contradiction avec la norme.

65 535 services au maximum peuvent écouter une seule adresse IP à un moment donné, du moins sans certaines astuces subtiles (cette limite ne risque pas d'avoir de conséquences désastreuses de sitôt).

Résumé de l'en-tête UDP

L'en-tête UDP illustré à la Figure 9-5 suit l'en-tête IP et précède les données véritables de l'utilisateur dans les paquets UDP. Il se compose de quelques champs : port source et port de destination (16 bits chacun), longueur du paquet, et somme de contrôle 16 bits chargée d'une vérification supplémentaire de l'intégrité du paquet reçu.

Et, maintenant, quelque chose de complètement différent.

Les paquets TCP

Le protocole TCP (*Transmission Control Protocol*), défini dans le RFC 793[5] et dont l'en-tête est illustré à la Figure 9.6, a pour but de fournir une méthode fiable de contrôle des erreurs lors de l'établissement d'une conversation entre deux systèmes. L'utilisation de TCP est plus appropriée qu'UDP pour toutes les applications, à l'exception de celles qui doivent échanger plus que des messages simples et courts.

Même si techniquement la connexion TCP (une couche virtuelle d'un point de vue de l'application) implémente des datagrammes IP distincts en traversant le réseau, elle autorise un mode de communication très proche de celui d'une conversation téléphonique ordinaire. Contrairement au trafic UDP, TCP garantit que le destinataire reçoit toujours les données telles qu'elles ont été envoyées (ou, si la récupération d'une erreur n'est pas possible, que la conversation est entièrement abandonnée). Dans des conditions normales, on peut également être sûr de l'identité de l'appelant, mais cet atout s'accompagne d'un coût plus élevé et d'une diminution des performances.

Avec le protocole TCP, les deux systèmes d'extrémité commencent par initialiser une connexion en utilisant un algorithme appelé poignée de main en trois temps. En règle générale, ils utilisent pour cela des paquets spéciaux vides (ils ne contiennent que les entêtes mais aucune charge), se mettent d'accord sur l'objectif, confirment leur identité à l'autre et s'entendent sur les numéros de séquence et d'acquittement initiaux. Ces numéros (un ensemble de valeurs 32 bits) assurent une transmission fiable et transparente, car ils sont incrémentés quand les données sont envoyées. Cela permet au destinataire de mettre en file d'attente les paquets entrants dans le bon ordre et de déterminer si une quelconque partie des données est manquante.

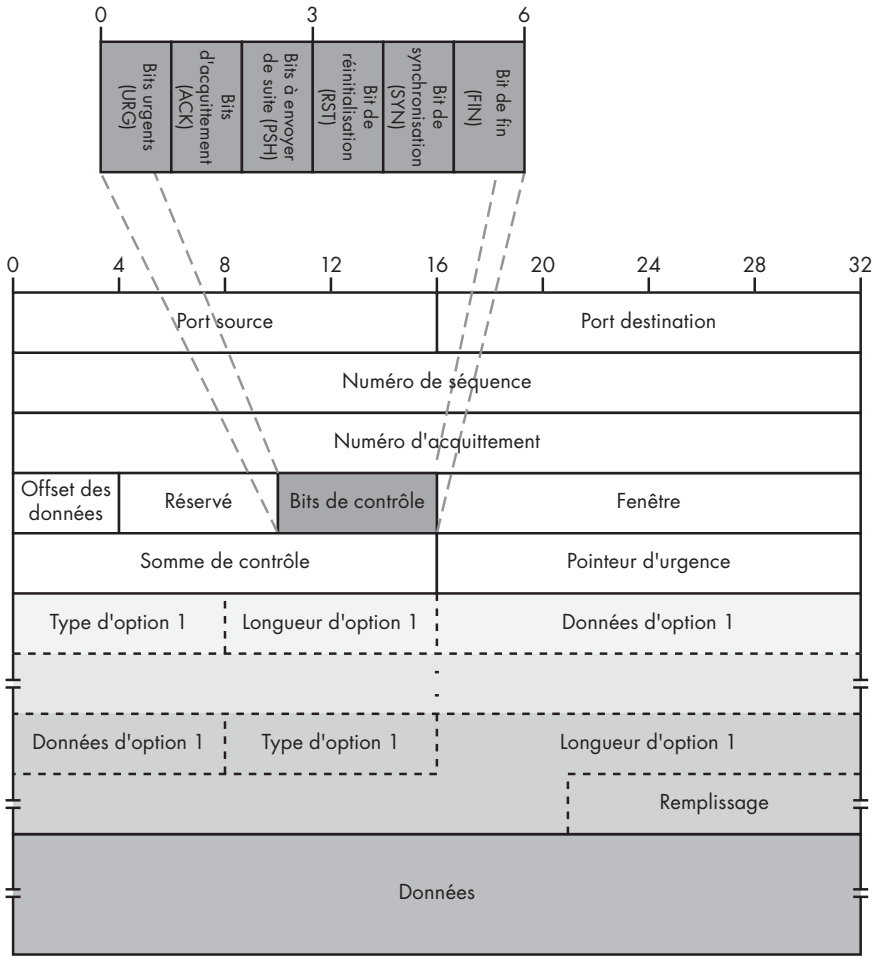


Figure 9.6

La structure de l'en-tête TCP.

Flags de contrôle : la poignée de main TCP

Une session TCP commence quand un système distant exprime le désir de se connecter à un port spécifique d'un ordinateur de destination. Le système distant envoie un paquet vide avec un flag SYN (autrement dit, un bit est défini dans l'en-tête) et un numéro de séquence initial dans son en-tête. Après réception de ce paquet, toutes les réponses doivent mentionner le numéro de séquence afin d'être honorées. Si l'ordinateur de destination n'envoie pas la bonne réponse dans un délai raisonnable, le paquet est envoyé à

nouveau, jusqu'à ce que la livraison réussisse ou que l'expéditeur décide que suffisamment de temps s'est écoulé et qu'il abandonne la connexion.

Ce numéro de séquence garantit que la réponse provient bien du destinataire, et non d'une personne tierce qui sait qu'une communication va être établie et qui cherche à l'intercepter. Le numéro de séquence assure également que la réponse est non pas un paquet égaré ou mal routé d'une session antérieure qui parvient finalement à destination mais bien une réponse à cette requête particulière de l'expéditeur (avec des numéros 32 bits et 4 294 967 296 valeurs possibles, le risque de collision est considérablement inférieur qu'avec les numéros 16 bits utilisés par les IP ID, ce qui minimise les risques d'erreur et rend tout à fait improbable qu'une personne extérieure parvienne à les deviner).

Le destinataire est censé répondre à une demande SYN avec un paquet similaire adressé à l'expéditeur sur le même port source. Ce paquet doit contenir un flag RST (là encore, un autre bit dans les en-têtes) pour indiquer qu'il n'est pas disposé à établir une session (aucun programme n'est prêt à répondre aux connexions sur ce terminal). Ce paquet doit également citer le numéro de séquence original dans sa réponse. Éventuellement, dans le cas peu courant où le destinataire souhaite réellement établir une connexion avec un inconnu, sa réponse doit être construite de la même façon mais comprendre les deux flags SYN et ACK pour indiquer qu'il accepte la requête. Il doit également inclure le numéro de séquence qu'il attend à partir de ce moment dans toutes les réponses appartenant à cette session.

Dans la dernière partie de la poignée de main, l'émetteur envoie un seul paquet ACK pour s'assurer que les deux parties connaissent les numéros de séquence et d'acquittement qu'elles se sont échangées précédemment et qu'elles sont synchronisées pour la transaction. En supposant que leur communication a atteint ce stade, les deux systèmes d'extrémité peuvent raisonnablement estimer que l'émetteur et le récepteur sont bien ce qu'ils prétendent être. Pourquoi ? Parce que chacun d'entre eux peut observer le trafic qui lui est adressé. Dans le cas contraire, si un système d'extrémité usurpait l'adresse IP afin d'établir une connexion au nom de quelqu'un d'autre, il n'aurait aucune idée du nombre à inclure dans sa réponse à l'autre partie (et cette dernière serait très surprise de voir que quelqu'un essaie de lui envoyer des paquets SYN+ACK ou ACK non sollicités).

Ce protocole de la poignée de main élimine le risque qu'une personne extérieure usurpe une identité mais il est toujours possible qu'une partie privilégiée et hostile se trouve sur un chemin légitime entre les systèmes (même si cela a moins de chance de se produire comparé au *spoofing* en aveugle).

NOTE

Même si le problème de l'utilisation de numéros de séquence difficiles à prédire n'était à l'origine pas considéré comme un problème alors que de nombreux systèmes utilisaient un simple générateur incrémentiel, la possibilité d'établir en aveugle une session en usurpant une poignée de main TCP depuis une source particulière ou en injectant des

données dans des connexions déjà établies est devenue un peu problématique avec le temps*. On considère maintenant comme nécessaire de sélectionner des numéros de séquence TCP initiaux avec soin afin qu'une machine témoin ne puisse pas prédire ce que sera la prochaine réponse du système à un paquet entrant, et plusieurs approches ont été élaborées pour faire face à ce problème[6].

Une fois qu'une poignée de main est terminée, les parties peuvent échanger des données, en reconnaissant mutuellement leurs numéros de séquence à chaque fois ; les paquets sur lesquels les numéros de séquence dépassent la taille de la "fenêtre" autorisée sont tout simplement ignorés. Ces numéros sont ensuite également régulièrement incrémentés pour tenir compte de la quantité de données envoyées jusque-là, ce qui permet de traiter les paquets reçus dans le bon ordre, même s'ils arrivent dans le désordre. Pour assurer la fiabilité, lorsqu'une partie des données n'est pas acquittée dans un délai raisonnable, une retransmission du paquet (ou des paquets) doit avoir lieu.

La fin d'une session survient lorsqu'un paquet FIN comprenant le bon numéro d'acquittement est reçu par l'une des parties. Si, à n'importe quel moment, l'un des systèmes veut brusquement mettre fin à la session (parce que, selon lui, il n'y a rien à communiquer, la session a expiré, ou qu'une partie a gravement violé la convention), un paquet RST est envoyé.

La Figure 9-7 (à gauche) illustre la réussite d'une poignée de main TCP légitime. La partie droite de la figure illustre l'échec d'une attaque par IP spoofing dans laquelle l'attaquant crée une session au nom d'une machine victime qui n'a pas l'intention d'échanger des données avec la cible. L'attaquant ne peut pas voir ou prévoir la réponse envoyée au système sur le compte duquel il essaie d'agir et ne peut donc pas terminer la poignée de main ni encore moins accomplir un réel échange de données au sein de la session TCP.

Comme indiqué, TCP fournit une protection raisonnable contre les problèmes de fiabilité du réseau et est donc plus adapté pour les communications ordonnées fondées sur les sessions. Mais la nécessité de mener à bien une poignée de main et l'obligation pour les deux terminaux de garder le contrôle des informations pour la connexion entraînent une surcharge supplémentaire. Le tribut à payer pour maintenir cet état est élevé car il devient nécessaire pour chaque connexion de suivre la trace des numéros de séquence, de l'état actuel du flux (phase de la poignée de main, phase d'échanges des données, phases de clôture), de conserver une copie de toutes les données envoyées mais pas encore acquittées au cas où il serait nécessaire de les renvoyer, et ainsi de suite.

* Kevin Mitnick, un des pirates au chapeau noir les plus célèbres et controversés, a compromis l'ordinateur de Tsutomu Shimomura en usurpant l'identité d'un de ses postes de travail, ce que M. Tsutomu a assez peu apprécié et ce qui a entraîné l'arrestation de Kevin.

En raison de leur coût sur la mémoire et les performances, les implémentations de pile TCP sont un vecteur d'attaque par déni de service courant.

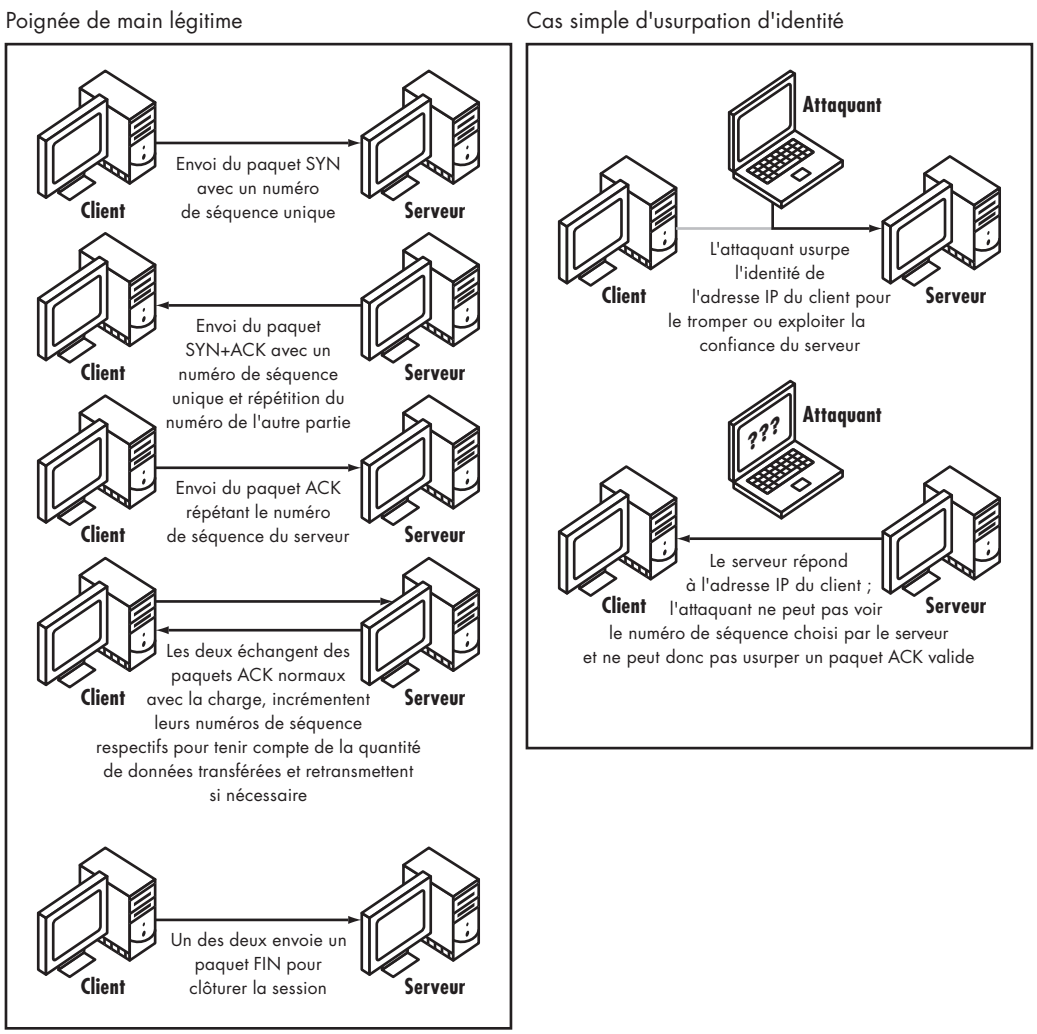

Poignée de main légitime
Client
Serveur
Envoi du paquet SYN avec un numéro de séquence unique
Envoi du paquet SYN+ACK avec un numéro de séquence unique et répétition du numéro de l'autre partie
Envoi du paquet ACK répétant le numéro de séquence du serveur
Les deux échangent des paquets ACK normaux avec la charge, incrémentent leurs numéros de séquence respectifs pour tenir compte de la quantité de données transférées et retransmettent si nécessaire
Un des deux envoie un paquet FIN pour clôturer la session
Cas simple d'usurpation d'identité
Attaquant
L'attaquant usurpe l'identité de l'adresse IP du client pour le tromper ou exploiter la confiance du serveur
???
Le serveur répond à l'adresse IP du client ; l'attaquant ne peut pas voir le numéro de séquence choisi par le serveur et ne peut donc pas usurper un paquet ACK valide

Figure 9.7

Une poignée de main TCP réussie et une tentative d'usurpation d'identité classique qui échoue.

Autres paramètres de l'en-tête TCP

D'autres paramètres TCP contrôlent également des aspects importants de l'interprétation et de la livraison de paquet. Ils nous seront utiles plus tard, quand nous essaierons d'obtenir des informations sur l'expéditeur en regardant juste les paquets de données qu'il fournit. La Figure 9-6, plus tôt dans ce chapitre, fournit la liste complète des champs TCP.

Ports source et de destination

Ces valeurs 16 bits identifient l'origine et la destination logique sur les machines source et de destination. Ils sont semblables aux paramètres des ports source et de destination utilisés par le protocole UDP, bien que l'espace des ports UDP et TCP soit séparé au niveau du système (autrement dit, une application peut écouter le trafic UDP sur le port 1234 tandis qu'une autre application écoute le même port dans l'espace TCP). La circulation est dirigée conformément à la spécification du protocole dans les en-têtes IP.

Les numéros de séquence et d'acquittement

Ces valeurs 32 bits assurent l'intégrité de la session. Un numéro de séquence est la valeur que l'émetteur attend que le destinataire lui renvoie. Un numéro d'acquittement est la valeur renvoyée à l'émetteur et n'aura de sens que si le flag ACK est défini.

Offset des données (à ne pas confondre avec le paramètre Offset du fragment IP)

L'information dans ce champ indique où l'en-tête se termine et où commence la charge dans le paquet. Comme pour les en-têtes IP, la longueur de l'en-tête TCP peut varier si certains paramètres de longueur variable lui ont été annexés. Cette information rend facile de passer directement aux données effectives, sans devoir relire toutes les informations d'en-tête.

Flags

Ces valeurs 8 bits définissent des propriétés spéciales d'un paquet. Chacun des bits désignés de ce champ représente un flag unique et peut être activé ou désactivé indépendamment ; en tant que tel, les flags TCP peuvent être arbitrairement recombinés. Les flags primaires (SYN, ACK, RST et FIN) définissent la manière dont le paquet doit être interprété dans une session TCP, comme indiqué ci-dessus. Les flags secondaires contrôlent certains aspects de la livraison de la charge et d'autres fonctionnalités étendues, comme la notification de congestion, mais ne sont pas utilisés pour modifier l'état de la connexion elle-même.

NOTE

Bien que les flags puissent être combinés à volonté, de nombreuses combinaisons possibles sont tout simplement illégales ou fausses (par exemple, SYN+RST n'a pas de sens et n'est pas autorisée d'un point de vue formel). Seules certaines combinaisons sont valables pour la poignée de main et le traitement normal des données. Comme les systèmes répondent différemment aux combinaisons illégales de flags, l'envoi de faux paquets comprenant des flags inhabituels est un mécanisme actif couramment utilisé pour détecter le système d'exploitation.

Taille de la fenêtre

Cette valeur 16 bits contrôle la quantité maximale de données qui peut être envoyée sans attendre un paquet d'acquittement. Une valeur plus élevée permet d'envoyer plus de données à la fois, sans avoir à attendre un accusé de réception, mais peut pénaliser les performances si une partie des données est perdue dans le transfert ou n'est pas acquittée et doit être de nouveau envoyée.

Somme de contrôle

Cette méthode 16 bits simple protège l'intégrité des données de la quatrième couche, semblable au mécanisme de somme de contrôle utilisé dans les en-têtes UDP et IP.

Pointeur d'urgence

Ce champ est interprété uniquement par le destinataire lorsque le flag secondaire URG est placé dans un paquet. Si URG n'est pas défini, la valeur spécifiée dans cette partie de l'en-tête est tout simplement ignorée. Ce flag indique que l'expéditeur demande au destinataire de relayer un message particulier à l'application qui gère le flux, probablement à la suite d'une situation "urgente", si bien que le paquet est inséré dans le flux logique à une position précédant celle qu'elle aurait dû normalement avoir ; l'offset exact est contrôlé par la valeur du pointeur de données urgentes. Ce mécanisme est rarement utilisé dans les communications normales.

Options TCP

Le bloc d'options de longueur variable situé à la fin de l'en-tête permet de spécifier des paramètres supplémentaires pour le paquet. Dans certains cas, il sera vide (longueur zéro), mais il est plus communément utilisé pour implémenter de nouvelles extensions du protocole qui ont été conçues plus tard, sans perturber les implémentations plus anciennes qui ne peuvent pas les comprendre. Le bloc d'options est conçu de telle sorte que les systèmes qui ne reconnaissent pas une option spécifique peuvent l'ignorer sans risque. Les options les plus populaires sont les suivantes.

Longueur maximale du segment (MSS)

Cette valeur 16 bits est égale à l'unité de transfert maximale sur le réseau de l'expéditeur, moins la taille des en-têtes de la couche inférieure. Elle représente la longueur maximale de paquet qui peut être renvoyée au destinataire sans entraîner de fragmentation en cours de route. L'expéditeur utilise le paramètre MSS pour garantir des performances optimales lorsque le destinataire retourne de grandes portions de données qui, dans le cas contraire, nécessiteraient d'être fragmentées et donc surchargeraient la bande passante. Malheureusement, l'option MSS est définie par le terminal en fonction de sa connaissance de la taille des paquets que les réseaux immédiatement voisins peuvent gérer ; il ne fait rien pour éviter le problème courant de la fragmentation qui se produit sur les systèmes intermédiaires (d'où la nécessité d'implémenter la découverte PMTU au niveau IP, comme nous l'avons dit précédemment).

Décalage de la fenêtre

Cette valeur de 8 bits décrite dans RFC1232[7] étend la portée du champ taille de fenêtre initialement défini dans l'en-tête TCP. L'expérience a montré que l'acquittement de chaque 64 kilo-octets de données (la valeur maximale du paramètre taille de la fenêtre de 16 bits) peut créer un goulet d'étranglement lors du transfert de grandes quantités de données, comme des fichiers multimédias, sur des liaisons haut débit dont la latence est élevée. Le redimensionnement de la fenêtre permet d'étendre la taille de la fenêtre et autorise donc l'envoi de plus de données sans attendre d'acquittement. Cela accélère le transfert des données mais peut aussi exiger que davantage de données soient retransmises lorsqu'un seul paquet est manquant.

Options d'acquittement sélectif (RFC2018[8])

En utilisant des tailles de fenêtre plus grandes, la perte d'un seul paquet nécessite la retransmission de l'ensemble du groupe de données qui ne sont pas encore acquittées, ce qui représente un terrible gaspillage de bande passante. Pour éviter cela, un mécanisme d'acquittement sélectif des portions de données a été conçu. Les terminaux commencent par déclarer leur capacité et leur volonté d'implémenter cette fonctionnalité en spécifiant une option d'acquittement sélectif SACK puis acquittent des blocs de données dans le désordre en utilisant l'option d'acquittement sélectif dans les en-têtes. L'implémentation de cette technique peut considérablement améliorer les performances, mais au prix d'une surcharge certaine de la mémoire et des opérations de traitement des données.

L'option Timestamp (deux valeurs 32 bits)

Il s'agit d'une autre extension de RFC1232 destinée à améliorer les performances. Ce mécanisme mesure le temps écoulé entre l'envoi et le retour des données (généralement choisi pour correspondre au temps du système ou à son uptime), ce qui permet à chaque système d'extrémité d'estimer le temps nécessaire aux allers-retours du flux. Le principal avantage de cette option est que l'expéditeur peut mesurer le temps typique dont un paquet a besoin pour atteindre sa destination et donc procéder à une retransmission TCP plus tôt s'il n'y a pas de réponse. L'option timestamp peut également être utilisée pour empêcher les collisions des numéros de séquence (PAWS, *Protection Against Wrapped Sequence* [Numbers]), par exemple lorsqu'un paquet envoyé depuis longtemps parvient à destination après que plusieurs gigaoctets de données ont été échangés et après le numéro de séquence.

EOL

Cette option doit être interprétée comme la fin des options ; elle indique au destinataire de ne pas traiter les données suivantes comme faisant partie de l'en-tête. Comme la taille de l'en-tête TCP est définie dans des unités plus longues qu'un seul octet, un espace inutilisé peut rester après avoir placé toutes les options avant le début des données, mais avant le début de la charge des données (ce qui n'est possible que sur une limite de 4 octets). L'option EOL peut être utilisée pour empêcher le destinataire de tenter d'analyser ces données.

L'option NOP

Cette option signifie "ne rien faire", et elle est tout simplement ignorée par le destinataire. L'expéditeur peut et doit utiliser des options NOP dans un paquet pour le remplir et assurer un bon alignement de certaines options multi-octets (qui doivent être alignées en raison des contraintes de performances et de l'architecture de certains processeurs*).

---

* "Doit" comme dans "nécessaire pour assurer le traitement approprié". Certains processeurs ont des pénalités de performances lors de l'accès à des structures de données multi-octets qui ne sont pas alignées sur 32 ou 64 bits ; d'autres ont besoin qu'elles soient alignées de cette manière, sinon ils provoquent une exception fatale (piège de l'exécution) et refusent d'effectuer une opération. Un émetteur malveillant peut placer délibérément des données mal alignées dans la mémoire tampon et espérer que le système du destinataire se bloque à la réception de ce paquet. Bien sûr, un système d'exploitation sain contrôle cela en premier ou essaie de copier les données de l'option dans une région correctement alignée avant de les traiter. Cependant, le bon sens d'un système ne doit pas être tenu pour acquis.

T/TCP (transaction TCP)

Cette extension ésotérique offre la prise en charge de sessions virtuelles distinctes (transactions) dans le cadre d'une session TCP établie. Cela permet d'éviter la surcharge provoquée par la nécessité de mener à bien une poignée de main chaque fois que vous voulez effectuer une opération particulière avec des services ponctuels (une approche qui est plus courante si une application veut effectuer le traitement d'un certain nombre de transactions distinctes avec un serveur. Cette extension est rarement utilisée, et elle est surtout utile pour certains systèmes de base de données (voir RFC1644[9]).

Les paquets ICMP (Internet Control Message)

Les paquets ICMP (voir RFC792[10]) sont utilisés pour envoyer des notifications et des informations de diagnostic pour d'autres types de protocoles. Logiquement considérés comme faisant partie de la couche trois, les paquets ICMP sont transportés comme une charge de paquets IP et, à ce titre, ne sont pas différents de la charge de la couche quatre. ICMP n'induit aucune nouvelle donnée de l'espace utilisateur entre les extrémités mais offre plutôt une méthode de signalisation simple pour le protocole IP. La Figure 9.8 montre la structure de l'en-tête ICMP.

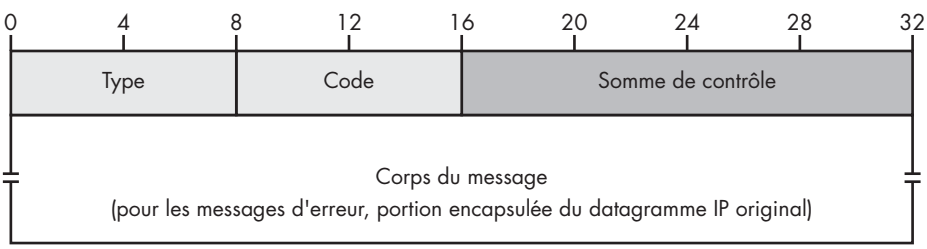


Figure 9.8

La structure de l'en-tête ICMP.

Différents messages sont envoyés en utilisant ICMP en réponse au trafic TCP ou UDP, généralement pour indiquer qu'un paquet en particulier ne peut pas être livré, que le transfert a expiré ou a été rejeté pour une raison quelconque. Plusieurs types d'ICMP peuvent être envoyés spontanément, comme des messages de routeur, des requêtes "écho" (ping), et ainsi de suite.

Comme pour les paquets UDP, l'en-tête ICMP est simple. Il se compose des champs suivants.

Type de message

Ce champ de 8 bits contient un message général sur la cause de l'envoi de ce paquet (comme "destinataire inaccessible"). Ce champ peut également porter un message indépendant, mais cet usage est peu fréquent.

Code

Cette valeur de 8 bits décrit le problème exact, si possible. Cela dépend du type de message et peut décrire l'état en détail ("le réseau n'est pas accessible", "la machine n'est pas accessible", "le port est inaccessible", "communication avec le réseau interdite"). Il n'est pas clairement indiqué quel niveau de détails doit être inclus dans le champ Type de message ou dans le champ Code.

Somme de contrôle du paquet

Ce champ vérifie que le paquet n'a pas été endommagé (comme dans UDP et TCP).

L'en-tête d'un paquet ICMP est assez simple et ne fournit pas de lui-même suffisamment d'informations pour résoudre le problème qu'il tente de signaler ou pour identifier le type de trafic qui a généré ce message. Cette information est transmise dans la charge du paquet et suit immédiatement l'en-tête d'un paquet.

Bien que la charge d'un paquet ICMP dépende du message, elle cite généralement le début du paquet qui a déclenché la réaction. Le destinataire peut ainsi déterminer à quelle communication correspond le message, et à quelle application indiquer ce problème. Il peut également l'utiliser pour garantir que l'expéditeur du paquet ICMP se trouve bien sur le chemin légitime du réseau entre les deux machines et non en dehors. Sinon l'expéditeur ne serait pas en mesure de voir les données effectivement échangées (il ne pourrait en particulier pas déterminer avec exactitude le numéro de séquence dans les paquets TCP). Cela permet d'éviter que des passants malveillants envoient de faux messages indiquant des problèmes de connectivité et de contraindre l'un des terminaux à abandonner une connexion, du moins en théorie. Naturellement, il peut être très difficile de distinguer le bon grain de l'ivraie, car certains systèmes sont connus pour altérer ou mal citer les données d'origine.

Le fingerprinting passif

Quel rapport la conception de ce protocole a-t-elle vis-à-vis de la vie privée des utilisateurs ? La réponse est un peu étrange : bien que la conception des paquets IP, TCP, UDP et ICMP soit généralement assez stricte et que les informations transmises dans ces entêtes ne soient pas particulièrement verbeuses, les différentes façons qu'utilise chaque système d'exploitation pour ajouter des informations à ces paquets permettent non seulement de connaître le type de système d'exploitation utilisé mais également la version spécifique de celui-ci sur une machine en particulier. Ces différences sont particulièrement évidentes dans le cas d'un flux qui n'est pas examiné clairement et de façon appropriée dans les spécifications ou qui n'est pas analysé dans les routines normales de vérification de la qualité des paquets (lorsque par exemple un des paquets comprend une combinaison illégale de flags comme SYN+RST).

Des recherches intensives destinées à différencier les systèmes en testant par saturation leur implémentation ont permis de conclure qu'il n'y avait pas deux implémentations IP identiques dans les systèmes d'exploitation. Il est souvent possible d'utiliser une analyse sophistiquée pour faire la distinction entre deux systèmes identiques fonctionnant sur des plates-formes légèrement différentes ou entre des versions légèrement différentes d'un système. Des outils d'analyse active comme le scanner de port NMAP, de Fyodor, qui permet d'obtenir une empreinte TCP/UDP ou le fingerprinting ICMP Xprobe d'Ofir Arkin, exploitent les défauts ou les bizarreries dans chaque système. Pour identifier le genre et la version du système d'exploitation, ils envoient différents types de paquets malformés ou inhabituels puis mesurent et analysent les réponses que ceux-ci déclenchent.

Examiner les paquets IP : les origines

Mais les techniques de prise d'empreintes du système ne s'arrêtent pas ici. En fait, saturer le système distant par l'envoi des données suspectes et facilement détectables constitue sans doute la manière la moins subtile d'aborder ce problème.

Au début de l'année 2000, deux personnes de Subterrain Security Group, connues seulement par leurs surnoms bind et aempirei, ont démontré qu'il est souvent possible d'obtenir des informations sur une entité distante sans procéder à aucune communication intrusive avec celle-ci ou, d'ailleurs, sans initier aucune communication (leur code et leurs découvertes ont d'abord été présentés lors de DefCON 8, une convention légèrement surestimée de hackers qui s'est tenue en 2000). Leur technique, aujourd'hui appelée *fingerprinting passif*, consiste à observer passivement (sic) le trafic légitime occasionnel en provenance d'un système distant. Bien que les mesures que cette technique utilise sont beaucoup plus subtiles et limitées que celles déployées par Fyodor et ses prédécesseurs, de nombreuses recherches (à laquelle je suis fier d'avoir apporté plusieurs contributions) ont fourni suffisamment d'observations pour atteindre un niveau assez étonnant de précision.

Pour mieux comprendre ce que l'on peut savoir à partir d'un seul paquet reçu sur le réseau, examinons les mesures sur lesquelles nous pouvons fonder le fingerprinting passif et ce qu'elles peuvent nous apprendre de l'autre partie. Pour cela, nous allons disséquer un des types les plus courants de trafics sur Internet : un paquet TCP légitime encapsulé dans un datagramme IP.

Time to Live initial (couche IP)

Rappelez-vous que le champ TTL contrôle le nombre de systèmes par lequel un paquet peut passer avant d'être considéré comme impossible à livrer et donc supprimé. La valeur TTL du paquet est diminuée à chaque fois qu'elle passe un routeur jusqu'à ce que le TTL atteigne zéro et que le paquet soit supprimé.

Comme la manière dont ce champ doit être fixé par l'expéditeur n'est soumise à aucune obligation stricte, de nombreux développeurs de pile IP se reposent sur le hasard pour définir la valeur par défaut de leur système favori. Même si un spectateur passif ne peut pas déterminer avec exactitude la valeur initiale du paquet sans effectuer de tests supplémentaires (car le paquet a sûrement traversé plusieurs routeurs avant d'être observé), il sait que cette valeur initiale doit être supérieure à l'état qu'il observe. En outre, la distance moyenne entre deux ordinateurs sur Internet ne dépasse pas quinze sauts en général, et il est rare que deux systèmes soient à plus de trente sauts de distance. Par conséquent, on peut sans risque supposer que la valeur d'origine se situe quelque part entre le TTL observé et le TTL observé + 30 (et qu'elle est inférieure à 256, bien sûr).

Comme nous connaissons les valeurs initiales utilisées par les systèmes d'exploitation les plus courants, nous pouvons savoir le genre du système d'exploitation que l'expéditeur est susceptible d'utiliser (Linux et les systèmes dérivés de BSD s'en tiennent habituellement à une valeur de 64 ; les développeurs Windows utilisent une valeur de 128, et certains vrais descendants d'Unix utilisent une valeur de 255). Ensuite, une fois défini le système d'exploitation qui a envoyé le paquet, en se fondant sur ce point et d'autres facteurs, nous pouvons aussi être en mesure de déterminer à quelle distance se trouve l'expéditeur du point d'observation, en soustrayant le TTL observé de la valeur connue pour avoir été utilisée au départ. En comparant cette valeur avec la distance à son réseau effectivement observée auparavant ou connue par d'autres moyens, nous pouvons alors tirer certaines conclusions sur l'organisation du réseau interne de l'expéditeur.

Le flag DF (couche IP)

Le flag DF dit : "Si la taille de ce paquet ne convient pas à une liaison réseau en particulier, ne pas la fragmenter, la supprimer." En observant si cette option est activée, il est possible de déterminer si le système utilise le mécanisme PMTUD décrit précédemment, ce qui donne encore une autre indication sur le système d'exploitation utilisé.

Cela établit également une distinction entre deux grands groupes de systèmes : seules les implémentations IP les plus récentes utilisent cette technique, et toutes les autres n'ont aucun intérêt à activer cette option dans les paquets qu'ils envoient.

Le numéro IP ID (couche IP)

Comme nous l'avons mentionné plus haut (au cours de la discussion consacrée aux imperfections de la fragmentation de paquet), certains systèmes où le PMTUD est activé définissent le numéro IP ID à zéro sur une partie (ou la totalité) du trafic sortant, car ils supposent que le trafic ne sera pas fragmenté et pour des raisons de sécurité liées à l'affichage des numéros d'identification IP (comme vous le verrez au Chapitre 13). Par conséquent, nous pouvons identifier ces systèmes en examinant si les paquets entrants ont un numéro d'identification IP fixé à zéro.

Toutefois, il existe un inconvénient. Bien que certains systèmes d'exploitation où le PMTUD est activé définissent toujours le numéro IP ID à zéro, d'autres systèmes peuvent également définir des identifiants IP à zéro à un moment donné, tout simplement parce que les numéros IP ID possibles ne sont pas si nombreux. En d'autres termes, si on voit un paquet dont le numéro d'identification IP est différent de zéro, on peut raisonnablement croire que ce n'est pas un système qui utilise des valeurs zéro pour toutes les communications sortantes. En revanche, si on voit une valeur zéro dans un paquet, il peut s'agir d'une espèce particulière de système où le PMTUD est activé mais tout aussi bien d'un système "régulier" qui a par hasard choisi zéro pour ce paquet.

Bien que cette probabilité soit faible, elle n'est pas non plus tout à fait négligeable. On peut soit ne pas accorder d'importance aux valeurs zéro des numéros d'identification (et ne considérer que les numéros IP ID différents de zéro pour réduire le nombre de systèmes d'exploitation possibles) soit procéder à plusieurs observations de la même source pour confirmer que les valeurs zéro sont toujours utilisées.

Type de service (couche IP)

De par sa conception, ce champ devrait être choisi pour indiquer aux systèmes intermédiaires quel est le type de trafic et quelle est la priorité du paquet, mais ce n'est presque jamais le cas. La plupart des systèmes d'exploitation donnent arbitrairement à ce champ une valeur fixe car les développeurs peuvent définir cette valeur comme ils le veulent sans affecter dans la pratique les opérations de mise en réseau TCP. Selon l'ego du développeur, il peut simplement donner à ce paramètre une valeur par défaut de zéro ou juger opportun de baliser toutes les communications provenant de son système comme ayant

une "latence faible", une "fiabilité élevée", ou tout autre paramètre en utilisant une combinaison de bits dans ce champ*.

Cela devrait nous donner un avantage car, en connaissant les valeurs par défaut de certains systèmes, nous pouvons une fois de plus réduire considérablement le nombre de systèmes possibles que l'expéditeur utilise. Pour ajouter à la confusion, toutefois, la valeur de ce champ est parfois modifiée pour tout le trafic sortant par certains vilains opérateurs DSL et autres fournisseurs d'accès Internet. Ils espèrent que certains routeurs distants de l'autre côté du globe tomberont dans le piège et considéreront que leur trafic, étiqueté comme "hautement prioritaire", mérite d'être accéléré et les traiteront avant d'autres connexions, ce qui devrait permettre aux clients de cette FAI de naviguer plus rapidement (cela est très douteux, en fait).

Comme pour les systèmes d'exploitation, une FAI choisit ses paramètres Type de service de façon plutôt arbitraire (par exemple, un fournisseur suédois utilise une combinaison assez unique et intéressante de bits de priorité définis à une valeur de 3 et de bits Type de service définis sur "un débit élevé"). Cette pratique, en retour, rend assez facile de détecter le trafic qui émane de certains fournisseurs d'accès Internet en repérant la sélection singulière qu'ils utilisent pour les bits Type de service, sans qu'il soit nécessaire d'effectuer une analyse active comme la recherche des registres WHOIS de l'adresse IP source.

Les champs Nonzero et Must Be Zero (couches IP et TCP)

Les spécifications IP et TCP demandent qu'un certain nombre de champs soient réservés pour une utilisation future. Tous les systèmes actuels devraient mettre ces champs à zéro, afin que les valeurs non nulles qui occupent ces positions dans un paquet puissent se voir attribuer une signification spéciale à l'avenir.

Inutile de dire que celles-ci ne sont pas à zéro dans certaines implémentations avant l'envoi, alors qu'elles devraient l'être. Ce problème n'est pas susceptible d'être considéré dans la phase de vérification de la qualité, car il ne cause aucun problème notable (les autres systèmes supposent qu'il vaut mieux prévenir que guérir et ne rejettent pas les paquets uniquement en raison de ce désagrément), si bien que ce défaut peut persister encore longtemps (peut-être jusqu'à ce que ces bits soient effectivement utilisés dans le cadre de certaines extensions TCP, ce qui entraînera un échec spectaculaire des communications avec ces systèmes défectueux). Une fois de plus, la capacité à examiner ces valeurs constitue une source précieuse d'informations pour identifier plus précisément le système d'exploitation de l'expéditeur.

---

* Certains développeurs choisissent même de régler le bit de ce paramètre avec la valeur Must Be Zero (doit être zéro), alors que cela ne devrait jamais être effectué dans une application légitime, sans doute juste pour faire une déclaration de style.

Port source (couche TCP)

Le port source identifie la partie à une connexion du côté de l'expéditeur. Chaque système affecte différemment les ports dits éphémères (émetteurs) pour les connexions sortantes, si bien qu'il est souvent possible de déterminer le système d'exploitation source en examinant le numéro du port observé. En outre, les systèmes utilisent couramment un éventail de ports spécifiques pour le *masquerading* (le masquerading, ou la traduction de plusieurs adresses réseau en une seule, implique la réécriture du trafic sortant d'un réseau privé afin que toutes les connexions semblent provenir du système de masquerading et que toutes les réponses soient retraduites et livrées à l'expéditeur réel lorsqu'elles sont reçues par le système).

Le masquerading est couramment utilisé dans les réseaux d'entreprises et domestiques pour préserver un espace d'adressage. Le réseau interne peut utiliser un grand nombre d'adresses qui, techniquement parlant, ne leur sont pas attribuées et qui ne sont pas routées là (ou ailleurs) depuis Internet. En revanche, les systèmes qui utilisent ces adresses peuvent toujours accéder à Internet en transmettant leurs connexions sortantes à un agent qui utilise sa propre adresse publique légitime pour atteindre le système distant au nom de l'émetteur. Cette approche protège également les systèmes internes, rendant impossible pour un étranger d'initier une connexion directe non sollicitée avec le système, tout en permettant aux seuls membres internes de se connecter à l'extérieur.

En examinant la plage de ports source choisie par l'autre partie, il est possible à la fois d'avoir une idée plus précise du système d'exploitation que l'expéditeur utilise et (une fois la plage mise en corrélation avec d'autres observations) de déterminer si l'expéditeur se trouve dans un réseau privé utilisant la traduction d'adresse (auquel cas la plage des ports source attendue pour ce système ne correspond probablement pas à celle effectivement observée). Si le réseau de l'expéditeur utilise la traduction d'adresse, il est également possible de tirer certaines conclusions sur le type de périphérique employé pour la traduction d'adresse, car chaque produit utilise des plages de ports différentes.

Taille de fenêtre (couche TCP)

Souvenez-vous que le paramètre taille de fenêtre détermine la quantité de données qui peut être envoyée sans acquittement. Ce paramètre spécifique est souvent choisi selon les règles personnelles et autres croyances du développeur. Dans les deux approches les plus populaires, cette valeur doit être un multiple du MTU moins les en-têtes de protocole (une valeur nommée MSS ou *Maximum Segment Size*) ou tout simplement quelque chose de suffisamment élevé et "arrondi". Les anciennes versions de Linux (2.0) utilisaient comme valeurs des puissances de 2 (16 384, par exemple). Linux 2.2 utilisait un multiple du MSS (onze ou vingt-deux fois la valeur du MSS, pour une raison

quelconque), et les nouvelles versions de Linux utilisent une valeur égale à deux ou quatre fois le MSS. La Dreamcast, une console de jeu disposant d'une connexion réseau, utilise une valeur de 4 096, et Windows utilise souvent 64 512.

Une application peut parfois modifier la valeur de la taille de fenêtre fixée par le système d'exploitation afin d'améliorer les performances, mais cela est rarement le cas (la présence d'une valeur qui ne correspond pas à la valeur par défaut d'un système d'exploitation représente un bon moyen de détecter une application spécifique ; Opera, un navigateur Web relativement courant, est un des rares exemples de ce type d'application).

Pointeur d'urgence et valeurs du numéro d'acquittement (couche TCP)

Les valeurs inscrites dans les champs du pointeur d'urgence (16 bits) et du numéro d'acquittement (32 bits) sont utilisées seulement si un flag TCP correspondant (URG ou ACK) est défini dans le paquet. Si ces flags ne sont pas définis, ces valeurs devraient être égales à zéro, mais elles ne le sont pas souvent. Certains systèmes les initialisent simplement à une valeur différente de zéro, ce qui n'entraîne pas de réel problème : comme ces valeurs ne seront pas interprétées si le flag approprié n'est pas défini, elles servent uniquement à identifier un système particulier.

Dans certains cas, cependant, ces valeurs ne sont pas initialisées du tout et sont simplement copiées à partir de ce qui se trouve à ce moment dans la mémoire tampon utilisée pour la construction du paquet TCP. J'ai observé ce comportement avec les implémentations de la pile sous Windows 2000 et XP en travaillant sur le fingerprinting passif du système d'exploitation : chaque fois que deux sessions TCP surviennent simultanément, ces valeurs contiennent certaines informations d'une session antérieure (un cas sur lequel nous reviendrons au Chapitre 11). Cela indique que la personne fait quelque chose d'autre à l'arrière-plan et révèle certaines informations transférées à une autre partie. Alléluia !

L'ordre des options (couche TCP)

Chaque système ordonne et sélectionne les options dans un paquet de façon unique. Comme il n'existe pas de règles régissant la manière dont les options doivent être ordonnées dans un paquet, il existe certaines combinaisons de "signature". Par exemple, Windows utilise une séquence caractéristique d'options "MSS, NOP, NOP, ACK sélectif autorisé" sur les paquets SYN tandis que Linux conserve généralement la séquence "MSS, ACK sélectif autorisé, Timestamp, NOP, Décalage de fenêtre". Naturellement, cela offre une fois de plus un excellent moyen de distinguer des systèmes.

Décalage de fenêtre (couche TCP, option)

Un facteur d'échelle pour la taille de la fenêtre est généralement défini à zéro. Toutefois, certains systèmes utilisent par défaut une valeur plus élevée ou augmentent continuellement ce paramètre pour un type précis de trafic lorsqu'ils concluent qu'il est raisonnable de le faire ; si par exemple l'utilisateur vient de télécharger un film piraté à partir d'un réseau P2P ou un fichier de grande taille de type différent (cette dernière hypothèse étant évidemment un peu moins probable).

Maximum Segment Size (TCP Layer, option)

Ce champ est fixé à une valeur spécifique sur certains systèmes. Sur d'autres, il indique le type de branchement direct au réseau du périphérique. Les différents types de réseaux ont des MTU différents, ce qui permet de dire si une personne utilise une connexion haut débit DSL ou un modem classique.

Données Timestamp (couche TCP, option)

Puisque cette valeur correspond souvent à l'uptime du système, il est souvent possible de le déterminer en observant l'option timestamp. En outre, pour un ensemble de systèmes d'exploitation donné, il est possible de les différencier et de suivre chacun d'eux en vérifiant les variations de timestamp dans le trafic entrant : des systèmes différents ont des uptime différents (et ont assez peu de chances d'avoir des temps de démarrage identiques), tandis que le même ordinateur maintiendra un accroissement continu de la valeur du paramètre timestamp.

Ceci devient très pratique dans deux situations. Cela est tout d'abord utile lorsqu'un ensemble de systèmes agit avec la même adresse IP, comme avec le masquerading. Dans ce cas, un webmaster curieux peut déterminer combien d'utilisateurs différents de la société X ont visité son site et sur quelle page se trouve chaque visiteur, même si toutes les demandes proviennent d'une seule adresse et semblent être indissociables au début.

Cela est ensuite pratique pour effectuer le suivi d'un utilisateur unique qui, pour quelque raison que ce soit, change fréquemment d'adresse IP. Pourquoi s'en soucier, et pourquoi vouloir déterminer si l'utilisateur le fait ? Ces utilisateurs peuvent par exemple basculer entre plusieurs adresses IP dynamiques affectées à une ligne d'accès réseau (en se déconnectant et se reconnectant) dans l'espoir que leurs tentatives d'attaque apparaissent comme un ensemble d'activités dénuées de sens et sans lien entre elles plutôt que comme une découverte du réseau exhaustive et bien planifiée. Ils peuvent également vouloir contourner les restrictions d'interaction sur un forum Web, pour un sondage en ligne ou un vote (histoire de bourrer les urnes), et ainsi de suite. Tout cela fait partie des passe-temps courants de la nouvelle génération.

La mesure de l'option timestamp est en général précise, car elle se base sur une horloge qui fonctionne le plus fréquemment à 100 ou 1 000 Hz (bien que certains systèmes utilisent 64 ou 1 024 Hz et des valeurs intermédiaires). Cette précision est suffisante pour différencier des boîtes même similaires qui ont presque toutes démarré en même temps après une panne électrique, et cette mesure fournit donc une précision extrême.

Autres types de fingerprinting passif

Dans ce chapitre, nous avons examiné les mesures les plus couramment utilisées pour déterminer le système d'exploitation d'un hôte distant (et suivre ses utilisateurs) à son insu. Mais, au-delà de ces éléments de base, de nombreux aspects intéressants et moins étudiés des communications peuvent être utilisés pour atteindre les mêmes buts, voire plus.

Par exemple, une variante intéressante de fingerprinting n'est pas liée à l'examen des paquets eux-mêmes mais consiste à mesurer le temps et les taux de réponse de certains messages ICMP, de certaines retransmissions TCP et d'autres caractéristiques similaires. Les valeurs utilisées pour tous les paramètres temporels (arrêt des transmissions et compteurs des retransmissions) fournissent un bon moyen d'obtenir une empreinte précise d'un système en particulier. Le projet CRONOS, fondé sur les études de Franck Veysset, d'Olivier Courtay et d'Olivier Heen, de l'équipe de recherche Intranode, vise à fournir un outil de fingerprinting actif fondé sur cet ensemble de mesures, mais les applications de fingerprinting passif sont tout aussi tentantes.

Une autre piste prometteuse consiste à combiner et à mesurer nombre d'autres anomalies ou paramètres rares, comme l'utilisation par l'expéditeur de certaines valeurs timestamp spécifiques, de numéros de séquence identiques aux numéros d'acquittement, de flags inhabituels, de données sur la charge dans les paquets de contrôle, de l'option d'EOL, et ainsi de suite. Ces caractéristiques peuvent également être utilisées pour distinguer les systèmes d'exploitation, bien qu'elles soient souvent spécifiques à un petit nombre d'implémentations (l'algorithme utilisé pour choisir les numéros de séquence initiaux est souvent une source d'information précieuse, comme vous le verrez au chapitre suivant).

Le fingerprinting passif en pratique

Ces mesures permettent d'identifier avec précision les systèmes d'exploitation, leur configuration et les paramètres réseau et également de suivre les utilisateurs de façon efficace et silencieusement. Même s'il peut sembler difficile de croire que cela est possible, p0f, un outil dont je suis l'auteur, met en œuvre la plupart de ces techniques pour recueillir et analyser les informations obtenues par l'analyse des paquets SYN,

SYN+ACK et RST d'une manière complètement passive et avec un taux de réussite élevé.

Prenons l'exemple d'un paquet pour voir l'efficacité de cette approche. Le tableau suivant contient un ensemble de paramètres importants extraits d'un paquet TCP réellement capté sur le réseau. Que peuvent-ils nous apprendre du système d'exploitation de l'expéditeur ?

Protocole IP (Version 4)	
Hôte source	nimue (10.3.0.1)
Hôte de destination	nightside (10.3.0.3)
Flags	DF
TTL	57
Numéro d'identification	4428
Options No IP (taille de paquet = 20)	
Protocole TCP	
Port source	3803
Port de destination	80 (HTTP)
Flags	SYN
Numéro de séquence	1418000073
Numéro d'acquittement	0
Taille de la fenêtre	32120
Options TCP	
1 MSS	1460
2 ACK sélectif autorisé	
3 Timestamp	170330930
4 Décalage de fenêtre	0

Voici ce que nous pouvons déduire de ces observations :

- Comme le flag DF dans les en-têtes IP est configuré, le système doit utiliser le PMTU. Les systèmes qui utilisent le PMTU sont les versions les plus récentes de

Linux, FreeBSD, OpenBSD, Solaris et de Windows. Nous pouvons écarter IRIX, AIX, de nombreux pare-feu commerciaux* et les autres systèmes qui n'implémentent pas PMTUD pour des raisons de fiabilité.

- Le temps de vie du paquet est de 57. Nous savons que la valeur initiale de TTL ne pouvait pas être inférieure car elle ne peut que diminuer au cours du transfert. En même temps, il est peu probable que cette valeur dépasse 87 (car alors le système serait vraiment très loin). Nous pouvons donc penser à beaucoup de systèmes Unix (qui tous utilisent un TTL initial de 64) et écarter Windows (avec lequel le TTL initial est de 128), les versions de Solaris antérieures à la version 8 (255) et plusieurs dispositifs réseaux (32).
- Le numéro d'identification du paquet est différent de zéro. Cela exclut Linux 2.4 et les versions plus récentes, ainsi que plusieurs versions récentes d'autres systèmes d'exploitation populaires.
- Le port source appartient à la gamme la plus couramment utilisée (1 024 à 4 095). Bien que cette indication ne suffise pas à elle seule à exclure des systèmes, on peut supposer que le système a établi plus de 2 700 connexions avant celle-ci et qu'il est peu probable qu'il soit derrière un serveur qui fait du masquerading.
- L'ordre des options (MSS, Selective ACK, Timestamp, taille de fenêtre) est spécifique à Linux 2.2 et versions ultérieures.
- La taille de la fenêtre (MSS*22) est un multiple de MSS. Le seul système qui corresponde est Linux 2.2.
- On n'observe aucune anomalie, ni de violations RFC, ni d'autres bizarreries dans le paquet, ce qui confirme l'hypothèse selon laquelle Linux est le système exécuté.
- La Longueur maximale du segment (MSS) indique un réseau Ethernet ou un modem utilisant le protocole de connexion PPP (MTU de 1 500).
- L'uptime du système est d'environ dix-neuf jours, et il est situé à sept systèmes du point d'observation.

Certes, certaines mesures peuvent être modifiées par des applications ou des réglages utilisateur (par exemple, les utilisateurs ont tendance à modifier le TTL et à activer ou à désactiver certains paramètres après la lecture de guides sur l'optimisation des réseaux ou l'exécution d'un "docteur du système"). Toutefois, par recoupement des observations, nous pouvons déterminer d'une manière fiable le système d'exploitation de la machine qui semble correspondre le mieux.

---

* Un pare-feu est essentiellement un routeur filtre, souvent aussi en mesure de comprendre et de prendre des décisions fondées sur les caractéristiques de la couche supérieure du flux.

Dans notre exemple, nous avons de bonnes raisons de croire que le système en question est Linux 2.2 et que l'expéditeur est connecté à Internet par Ethernet ou par modem téléphonique. À partir de cette hypothèse, nous pouvons aussi conclure que le système se trouve à sept sauts (64-57, 64 étant le TTL initial pour les systèmes Linux) et que son uptime est proche de vingt jours. Si plusieurs utilisateurs se cachent derrière cette IP particulière, on peut facilement les compter et distinguer leurs sessions en fonction des caractéristiques de leurs systèmes et des données de timestamp, si elles sont disponibles.

Les applications de fingerprinting passif

En observant le trafic du réseau, le destinataire ou un intermédiaire (comme un FAI entre l'expéditeur et le destinataire) peut obtenir des informations qui dépassent le cadre des données effectivement échangées, y compris certains paramètres du système de l'expéditeur. Comme suggéré précédemment, l'exposition est importante et très intéressante car, contrairement aux données transmises par les applications, elle n'est pas forcément évidente et échappe souvent au contrôle de l'utilisateur. Bien que les utilisateurs puissent modifier les paramètres de leur navigateur et ceux d'autres applications afin d'empêcher d'être surveillés, identifiés et suivis, les informations divulguées par la couche inférieure IP ou TCP peuvent facilement compromettre cet effort, en dévoilant à l'observateur autant de choses sur la victime que celle-ci essaie de cacher. Cette couche peut également transporter des données d'une importance plus fondamentale pour la sécurité de l'infrastructure, notamment des indications utiles sur la façon dont le réseau de la victime est construit et protégé.

Ceci dit, si l'on excepte l'invasion de la vie privée, le fingerprinting passif peut aussi être utilisé pour des tâches de reconnaissance tout à fait légitimes. Toutes les applications pratiques (et couramment déployées) de fingerprinting passif peuvent aussi bien être utilisées dans un but malveillant que pour se défendre de façon légitime.

Collecter des données statistiques et des incidents de connexion

Une des utilisations légitimes du fingerprinting passif consiste à surveiller le réseau pour effectuer une analyse objective et non intrusive des plates-formes et des environnements réseaux utilisés, afin de garantir que les utilisateurs reçoivent un service optimisé pour leurs logiciels et qu'aucun groupe d'utilisateurs n'est négligé d'une façon ou d'une autre. De plus, le fingerprinting passif peut grandement faciliter la collecte d'information sur des agresseurs potentiels ou sur toute autre activité non autorisée. Il est en particulier très populaire lorsqu'on utilise des pots de miel.

NOTE

Le pot de miel est un concept que Lance Spitzner, de Sun Microsystems[11], étudie et dont il fait l'apologie. L'objectif est de laisser le propriétaire se renseigner sur ses opposants et leurs objectifs, en utilisant des périphériques (pots de miel) vulnérables qui n'ont pas de réelle importance pour l'infrastructure mais qui sont conçus pour sembler en avoir une. Ils sont intéressants dans le sens où ils apprennent au responsable de la sécurité comment les attaquants agissent de façon non autorisée et illicite.

Optimisation du contenu

Une application active du fingerprinting passif repose sur la fourniture de services optimisés à un destinataire spécifique à partir d'une analyse de la configuration qu'il utilise pour accéder au serveur. Je considère qu'il est de mon devoir de citer de nouveau un de mes outils : p0f. p0f offre une méthode d'interrogation sur les paramètres des connexions entrantes récentes à partir d'autres applications, ce qui rend l'optimisation du contenu beaucoup plus facile : un script Web n'a pas besoin de savoir beaucoup de choses sur TCP et IP, il peut simplement demander à p0f "qui est ce gars à qui je parle ?" et obtenir une réponse utile.

Élaboration d'une politique

La détection et le blocage éventuel des installations obsolètes ou des systèmes non conformes (par exemple les périphériques qui violent la politique de l'entreprise ou posent un risque de sécurité) ou les liaisons réseau non autorisées sont d'autres applications intéressantes du fingerprinting passif. Depuis la version 3.4, OpenBSD fournit une méthode pour router et rediriger le trafic en fonction des résultats obtenus par la détection du système d'exploitation, ce qui rend tout à fait viable l'élaboration d'une politique fondée sur les caractéristiques du système d'exploitation distant. La même fonctionnalité est désormais fournie par le code patch-o-matic dans Linux netfilter. Les deux implémentations sont étroitement inspirées par p0f ou fondées dessus.

La sécurité du pauvre

Le fingerprinting passif peut aussi être utilisé pour minimiser plusieurs sortes d'expositions. Bien qu'il soit possible de tromper la technique de fingerprinting avec divers efforts, il pourrait être utilisé pour empêcher certains types de clients (comme les systèmes Windows, une plate-forme plus fréquemment infestée par des logiciels espions, des portes dérobées et des vers qui est souvent utilisée pour la distribution en masse d'e-mails non sollicités ou pour une attaque basée sur le nombre de sauts) d'utiliser nombre de services sous-jacents sur le réseau, tout en permettant aux entités les "moins suspectes" d'y accéder.

Test de sécurité et découverte du réseau

Le fingerprinting actif est souvent stoppé dès le départ par les pare-feu et d'autres solutions qui filtrent et analysent attentivement le trafic IP. Le fingerprinting passif, en revanche, peut étudier des systèmes même très bien protégés et établir une carte des réseaux sans déclencher aucune alerte.

L'utilisation du fingerprinting passif pour tester et évaluer la sécurité est double. Premièrement, il peut être utilisé pour analyser le trafic entrant. Même si l'observateur doit attendre que le parti distant se connecte à ses systèmes, une telle connexion peut être assez facilement initiée sans déclencher de soupçon. En fait, il suffit souvent d'envoyer un e-mail ou un lien particulier vers un site Web à la victime, même si elle se trouve derrière la solution la plus sophistiquée de filtrage des paquets. Deuxièmement, le fingerprinting passif peut être utilisé pour analyser les réponses légitimes à un service disponible afin de déterminer les paramètres de la partie distante. Si un hacker à chapeau noir sait comment compromettre un réseau interne mais souhaite en savoir plus sur son fonctionnement interne afin de réduire au minimum le risque d'être détecté prématurément, le fingerprinting passif peut se révéler utile. On peut en dire autant des tests de sécurité légitimes pour celui qui est payé par l'entité qui subit le test.

Profiling du consommateur et invasion de la vie privée

De nombreuses entreprises se donnent beaucoup de mal pour rassembler et vendre des informations précieuses sur les habitudes, les préférences et les comportements de la population. Bien que cette information soit souvent utilisée à des fins commerciales, elle pourrait en théorie être utilisée contre une personne en particulier. La capacité de suivi des utilisateurs en mettant en corrélation les résultats du fingerprinting passif pour les différents sites qu'ils ont visités (que ce soit pour établir la carte des réseaux interne et des logiciels utilisés, pour suivre les individus ou pour recueillir d'autres données statistiques précieuses) peut être une source d'informations susceptible d'avoir une valeur considérable en elle-même ou d'être utilisée pour renforcer l'attrait d'autres offres qui ne respectent pas forcément l'éthique.

Espionnage et reconnaissance furtive

Il est souvent très tentant de recueillir des informations supplémentaires sur l'architecture du réseau d'un concurrent et sur le comportement et les préférences d'un utilisateur. Bien que cela puisse relever de la mauvaise science-fiction, il s'agit simplement d'un type plus ciblé de profiling que celui abordé ci-dessus.

Protection contre le fingerprinting

Étant donné la complexité d'une pile IP typique, il est extrêmement difficile d'empêcher le fingerprinting en général, mais il est possible de résoudre certains problèmes spécifiques et de désactiver certains types de logiciels de fingerprinting connus, en déterminant sur quel paramètre il s'appuie le plus et en le changeant ensuite. Par exemple, certaines solutions de filtrage de paquets, comme pf dans OpenBSD, fournissent un service de normalisation des paquets qui assure que tout le trafic sortant "ait la même apparence". Même si cela peut dans une certaine mesure contrer certains aspects du fingerprinting ou le rendre simplement plus difficile en diminuant la précision de certains programmes populaires, cela ne résout pas complètement le problème.

Bien que la modification manuelle ou automatisée, approfondie et apparemment exhaustive de plusieurs réglages du système d'exploitation ou de certains paramètres TCP puisse rendre plus difficile l'identification du système, certains comportements sont profondément enfouis dans le noyau et ne sont pas personnalisables. Par exemple, il est assez difficile de changer l'option d'ordonnancement d'un paquet. En outre, lorsque les utilisateurs effectuent des modifications manuelles, ils risquent d'introduire des caractéristiques uniques dans les paquets provenant de leur système, ce qui a pour seul résultat d'affecter encore plus leur vie privée et leur anonymat.

Heureusement, des solutions répondent à certains types de tests. Par exemple, IP Personality, de Gaël Roualland et de Jean-Marc Saffroy, modifie la pile TCP afin qu'elle semble provenir d'un autre système d'exploitation pour certains outils. Si vous le souhaitez, vous pouvez utiliser IP Personality pour faire penser à NMAP que votre système est une imprimante laser Hewlett-Packard. Toutefois, divers problèmes se posent. Tout d'abord, il est facile d'affaiblir réellement la pile TCP d'un système en usurpant l'identité d'un périphérique qui utilise une pile faible. Si par exemple vous devez utiliser des numéros de séquence simples sur toutes les connexions afin de vous conformer aux caractéristiques particulières d'une imprimante, quelqu'un profitera tôt ou tard de cet avantage pour facilement perturber ou modifier votre trafic. En outre, des logiciels comme IP Personality ne fonctionnent que pour les outils les plus populaires, célèbres et bien documentés, mais n'offrent aucune garantie de succès contre les autres, car les caractéristiques que chaque outil examine et la manière dont il les interprète varient. Vous ne pouvez qu'espérer tromper l'attaquant le moins déterminé, le plus naïf et le plus "grand public"qui utilise les outils que vous connaissez.

NOTE

Contrairement aux agents de masquerading, un pare-feu de type proxy et les autres dispositifs de proxy ne transfèrent pas les paquets mais interceptent les connexions et en initient de nouvelles en utilisant leur propre IP. Il s'agit de la seule solution complète au

fingerprinting des troisième et quatrième couches OSI, mais ils ont un grave impact sur les performances et sont plus sujets aux problèmes en raison de la plus grande complexité qu'ils apportent. En outre, un fingerprinting de plus haut niveau de l'application elle-même est toujours possible.

Matière à réflexion : le défaut fatal de la fragmentation IP

En examinant les caractéristiques qui définissent le protocole IP, j'ai fortuitement indiqué que le processus de fragmentation et de réassemblage de paquets présentait des défauts fatals. Cette idée m'est venue principalement en raison d'une observation assez intéressante que j'ai faite lors de l'écriture de ce livre. Bien que ce concept soit lié à une attaque active et visible exécutée par une entité ouvertement malveillante (même s'il n'est pas facile de remonter jusqu'à cette entité), il s'agit d'un défaut unique et intéressant qui est inhérent à la conception du protocole IP. C'est non pas le résultat d'une erreur clairement définie, mais plus une collision de paradigmes sur les différentes couches de la conception d'ailleurs curieusement définis par Jon Postel, l'un des concepteurs du protocole IP. J'ai décidé de l'inclure ici pour clore ce chapitre, comme matière à réflexion pour ceux qui s'intéressent à la pathologie des failles informatiques.

Examinons d'abord l'état des choses, de nos jours ou peut-être hier, en revenant sur une assez vieille technique d'attaque déjà mentionnée lors de la discussion portant sur le protocole TCP. La technique en question, le *blind spoofing*, a été décrite pour la première fois par Robert T. Morris au milieu des années 1980[11]. Elle a connu sa période de gloire une décennie plus tard, mais son importance a diminué depuis. Nous allons nous concentrer sur un exemple précis *d'usurpation d'identité aveugle*, celui qui consiste à injecter certaines données dans une session existante afin de la perturber, de convaincre le serveur que l'utilisateur a émis une commande particulière ou encore de convaincre l'utilisateur qu'il reçoit une réponse spécifique du serveur. Cette technique est souvent appelée *détournement de connexion* (*connection hijacking*).

Dans des circonstances normales, une machine témoin malveillante voulant insérer des données dans un flux TCP doit d'abord déterminer les numéros de séquence utilisés par au moins l'une des parties. Même si une telle attaque est très dépendante du temps et doit être ciblée contre une connexion existante en particulier, elle peut être (et a été à maintes reprises) exécutée avec succès lorsque les numéros de séquence sont prévisibles. En fait, à la fin des années 1990, de nombreux outils ont été utilisés pour perturber les sessions TCP de Windows sur les réseaux IRC (*Internet Relay Chat*) en exploitant la faiblesse de l'algorithme de sélection du numéro de séquence initial (ISN) dans

Windows. Il suffisait d'injecter un seul paquet RST ici et là pour éjecter une personne hors du serveur. C'est ce que nous appelions s'amuser à l'époque.

De nos jours, la situation est un peu différente. Grâce aux efforts de nombreux chercheurs (y compris l'humble auteur de ces mots), les développeurs ont travaillé d'arrache-pied pour rendre les numéros de séquence initiaux des connexions TCP plus difficiles à prévoir. De nombreuses tentatives pour améliorer la qualité et la robustesse des générateurs de numéro de séquence dans les systèmes d'exploitation courants ont fini par rendre les attaques par prédiction de l'ISN plus difficiles, à quelques exceptions près qu'il est inutile de citer. Les systèmes qui utilisent des numéros ISN séquentiels ont en grande partie disparu ; un attaquant dans l'incapacité de déterminer les nombres utilisés dans une communication avec une autre partie est forcé d'effectuer une recherche des valeurs possibles dans tout l'espace 32 bits pour effectuer une attaque par insertion de données précise (moins s'il souhaite simplement interrompre ou altérer irrévocablement la session). Cela représente environ 4 294 967 296 combinaisons, si bien que cette attaque demande à l'attaquant d'envoyer environ 80 Go de données pour qu'elle réussisse. Inutile de dire que ce n'est pas considéré comme particulièrement réalisable.

Toutefois, en ce qui concerne les avantages réels que procure une attaque par injection de données réussie, peu de choses ont changé. Même si de plus en plus de communications sont échangées sur des couches qui prennent en charge le cryptage, la pertinence de ce type d'attaque n'a pas été sensiblement réduite ; beaucoup de scénarios d'attaque fructueuse subsistent. En voici quelques exemples :

- Les données peuvent être insérées dans un transfert non crypté de serveur à serveur ou de routeur à routeur, comme c'est le cas des échanges d'e-mails, des transferts de zones DNS, des communications BGP, et ainsi de suite. Une grande partie du trafic de serveur à serveur peut être générée par l'attaquant et pourtant contenir des informations sensibles ou de confiance, ce qui rend une attaque ciblée et fondée sur le temps plus réalisable.
- Les données peuvent être insérées dans un transfert non crypté de client à serveur, comme le téléchargement de fichier FTP (*File Transfer Protocol*) ou les réponses HTTP (*Hypertext Transfer Protocol*). Cette attaque peut être effectuée afin que des éléments malveillants, compromettants ou diffamatoires soient fournis à un visiteur d'un serveur de plus haut niveau ou pour faire croire qu'un visiteur innocent est l'auteur d'une tentative de compromission.
- Les données peuvent être insérées dans une session existante pour exploiter une vulnérabilité dans le service à un stade qui n'est pas disponible à un utilisateur qui ne s'est pas authentifié. Cela vaut aussi bien pour le trafic chiffré ou non. Par exemple, un service comme POP3 (*Post Office Protocol Version 3*, un protocole d'accès à distance de sa boîte aux lettres électronique) peut accepter différentes commandes uniquement si l'utilisateur a réussi à se connecter. Avant la connexion, les seules

commandes disponibles sont celles qui concernent directement le processus d'authentification (directives USER et PASS). Sans un mot de passe valide, l'attaquant ne peut pas exploiter une faille dans une des commandes disponibles plus tard (comme RETR, une commande utilisée pour récupérer un message en particulier d'une boîte aux lettres électronique). Cependant, si l'attaquant parvient à injecter une requête RETR malveillante dans la session existante d'un utilisateur déjà authentifié, il a gagné.

- Même un flux sécurisé et crypté et dont l'intégrité est protégée peut subir une attaque par déni de service lorsqu'une session est perturbée ou abrogée par un seul paquet soigneusement préparé.

À ce titre, il est tentant d'être en mesure d'injecter des données avec un minimum d'effort, sans avoir à chercher tous les numéros de séquence possibles. Et c'est là que la fragmentation devient très pratique.

Fragmentation TCP

Quand un paquet IP transportant une charge TCP est fragmenté (un phénomène courant au cours du transfert des fichiers que l'utilisation d'un flag DF par certains systèmes n'empêche pas toujours), les données voyagent dans le réseau en plusieurs morceaux et sont réassemblées seulement quand elles arrivent au destinataire. Un attaquant intelligent, en anticipant cette fragmentation, peut envoyer un fragment IP illégitime spécialement conçu et le faire passer pour un paquet de l'expéditeur attendu. À la réception de ce fragment, le destinataire peut, avec un peu de chance (une question de temps exact), finir par l'utiliser au lieu du vrai fragment pour réassembler le paquet initial.

Dans ce scénario d'attaque, le premier fragment (contenant l'intégralité des en-têtes TCP, y compris les ports exacts, les numéros de séquence, et ainsi de suite) est fusionné avec une charge malveillante falsifiée par l'attaquant. En conséquence, l'attaquant n'a pas besoin de connaître les numéros de séquence ou d'autres paramètres de la session pour insérer ses données dans la trame, sapant de ce fait l'ensemble des efforts de génération ISN. Une fois l'attaque terminée, le paquet final traité par le destinataire se compose des données d'en-tête valide copiées à partir d'un fragment légitime et d'une charge malveillante injectée par l'attaquant.

NOTE

L'attaquant peut remplacer une partie de la charge dans le premier fragment en spécifiant un léger chevauchement entre les fragments ; de nombreux systèmes honorent les chevauchements entre les fragments et écrasent les données reçues auparavant avec celles de la nouvelle copie. Dans le cas le plus extrême, l'attaquant peut parvenir à remplacer toutes les données à l'intérieur d'un paquet TCP sauf le numéro de séquence.

Naturellement, certaines pièces du puzzle manquent toujours. Dans ce scénario, l'attaquant a non seulement besoin d'agir à un instant précis et de savoir quand une transmission a lieu*, mais il doit surmonter deux obstacles :

- Les fragments doivent avoir un numéro IP ID correct pour que le fragment de l'attaquant soit fusionné. Heureusement, ce n'est pas un problème sur de nombreux systèmes, car les identifiants IP sont choisis de façon séquentielle. Ainsi, le numéro susceptible d'être utilisé à ce moment peut être déduit en tentant simplement un test de connexion. Certains systèmes, notamment Linux, FreeBSD et Solaris, offrent des numéros d'identification aléatoires, ce qui peut rendre plus difficile l'attaque mais pas l'empêcher. L'attaquant doit tout simplement vérifier des milliers (et non des milliards) de combinaisons, car le champ IP ID est assez court (2 octets seulement).
- L'en-tête TCP contient une somme de contrôle qui est vérifiée après le réassemblage, si bien que la somme de contrôle des données modifiée par l'attaquant doit être la même que celle de la charge originale. Toutefois, étant donné que la conception d'une somme de contrôle TCP est simple (une variation d'une somme de 16 bits), on peut forger une charge qui ne modifie pas la somme de contrôle, tant que la section originale à remplacer est connue de l'attaquant (ce qui est souvent le cas, en particulier pendant les transferts de fichiers, lorsque l'attaquant veut insérer du code ou du contenu malveillant dans une partie des données disponible publiquement).

L'exemple suivant de somme de contrôle simplifiée d'un paquet qui est formée des mots d'en-tête H1 et H2 et des mots de charge P1, P2 et P3 illustre ceci :

```
C = H1 + H2 ... + P1 + P2 + P3 ...
```

H1, H2 et C ne sont pas connus de l'attaquant (les en-têtes contiennent des numéros de séquence et la somme de contrôle est affectée par ces données). L'attaquant n'a aucun moyen d'examiner effectivement ce paquet, mais il sait que la victime effectue une transaction spécifique (prévisible) au niveau de l'application (il contrôle par exemple ses e-mails, télécharge une page Web, discute avec des amis, etc.). L'attaquant peut déduire les données P1, P1 et P3 de la charge et vouloir les remplacer par ses propres mots malveillants N1 et N2, en utilisant un troisième mot pour la compensation de la somme de contrôle (CC) de sorte que le paquet reste valide.

```
C = H1 + H2 ... + N1 + N2 + CC ...
```

---

* Le timing lui-même n'est pas un problème si grave qu'il peut sembler l'être à première vue. L'attaquant peut choisir d'envoyer son deuxième fragment malveillant un peu en avance ; le destinataire crée alors une mémoire tampon de réassemblage et attend pendant une certaine période de temps les portions restantes. Une fois que le premier fragment légitime arrive, le contenu de la mémoire tampon est considéré comme totalement reconstruit, sans attendre que le deuxième morceau véritable n'arrive.

En résolvant ces équations pour CC, on peut conclure que la somme de contrôle doit être compensée avec CC = (P1 + P2 + P3 – N1 – N2). L'attaquant peut alors modifier le paquet pour que la somme de contrôle reste la même sans connaître l'ensemble du paquet ; il n'a besoin que du bit de remplacement. Cela est suffisant pour calculer correctement le bit de compensation et préserver la somme de contrôle.

10

Stratégies avancées pour compter les moutons

Où l'on examine comment déterminer l'architecture du réseau et localiser l'ordinateur.

Effectuer la reconnaissance et établir la cartographie du réseau consistent à exploiter un ensemble de vecteurs de divulgation d'informations inhérents au principe des protocoles de communication Internet afin de reconnaître les systèmes et les réseaux ou d'identifier ou de suivre des attaquants potentiels, des utilisateurs, des clients ou des concurrents. Il s'agit sans doute de l'analyse des données passives la plus développée, la plus largement déployée, la plus importante et la plus utile à ce jour. Cependant, certains problèmes affectent dans certains cas à la fois sa précision et sa facilité d'utilisation, en particulier pour les techniques de fingerprinting passif TCP/IP les plus connues et éprouvées.

Les avantages et les implications du fingerprinting passif classique

Les mesures de fingerprinting passif examinées au chapitre précédent permettent d'identifier facilement certaines caractéristiques provenant d'un système ou d'un réseau. Dans plusieurs cas, ces techniques permettent également de suivre la trace des utilisateurs lorsqu'ils changent d'adresse ou la partagent avec d'autres sur un même réseau. On peut utiliser ces techniques sans interagir avec la partie distante aussi longtemps qu'on peut persuader la personne observée d'interagir avec un réseau spécifique ou tant que ses communications réseau passent par un ensemble spécifique de systèmes contrôlés par une personne suffisamment curieuse. Ainsi, le fingerprinting passif permet notamment au propriétaire d'un serveur ou à un fournisseur d'accès d'acquérir nombre d'informations assez facilement et de façon totalement furtive.

Le fingerprinting passif fournit une arme à double tranchant à la partie distante. On peut le déployer pour obtenir des données utiles sur la structure interne d'un réseau, afin de rendre plus facile une attaque ou pour en apprendre davantage sur les technologies de gestion de réseau utilisées (même dans un environnement assez complexe, comme le montre la Figure 10.1). On peut aussi l'utiliser (à juste titre) pour surveiller les violations éventuelles sur son propre réseau (par exemple des connections ou des points d'accès illégaux qui connectent un réseau interne au monde extérieur) ou encore pour suivre la trace des attaquants.

La perte de confidentialité qui en résulte pour un utilisateur unique est généralement négligeable, à moins qu'établir un lien entre les activités occasionnelles d'un utilisateur avec les données acquises par le fingerprinting ou qu'effectuer le suivi d'un seul utilisateur à travers plusieurs domaines pose un problème particulier (ce qui a des chances de se révéler exact seulement lorsque le comportement de l'utilisateur est contestable). La perte cumulée de confidentialité pour tous les utilisateurs peut devenir très inquiétante, mais les informations recueillies au moyen du fingerprinting ou le suivi effectué à l'aide du fingerprinting peuvent avoir une valeur marchande considérable (vos données personnelles peuvent être vendues plus cher aux annonceurs si elle s'accompagnent d'informations sur vos préférences et vos centres d'intérêt, par exemple). L'exposition du fonctionnement technique interne d'un réseau peut également être indésirable pour les entreprises et d'autres types d'infrastructures sensibles.

Néanmoins, tout n'est pas encore perdu. Comme indiqué précédemment, certains problèmes se posent pour obtenir des résultats précis à l'aide du fingerprinting passif. Le manque de fiabilité de la technique traditionnelle de fingerprinting passif du système d'exploitation montre à quel point il est facile de tromper l'observateur en réglant soigneusement certains ou la totalité des paramètres réseau utilisés sur un système qui est observé. Même s'il n'est pas particulièrement facile de modifier complètement tous

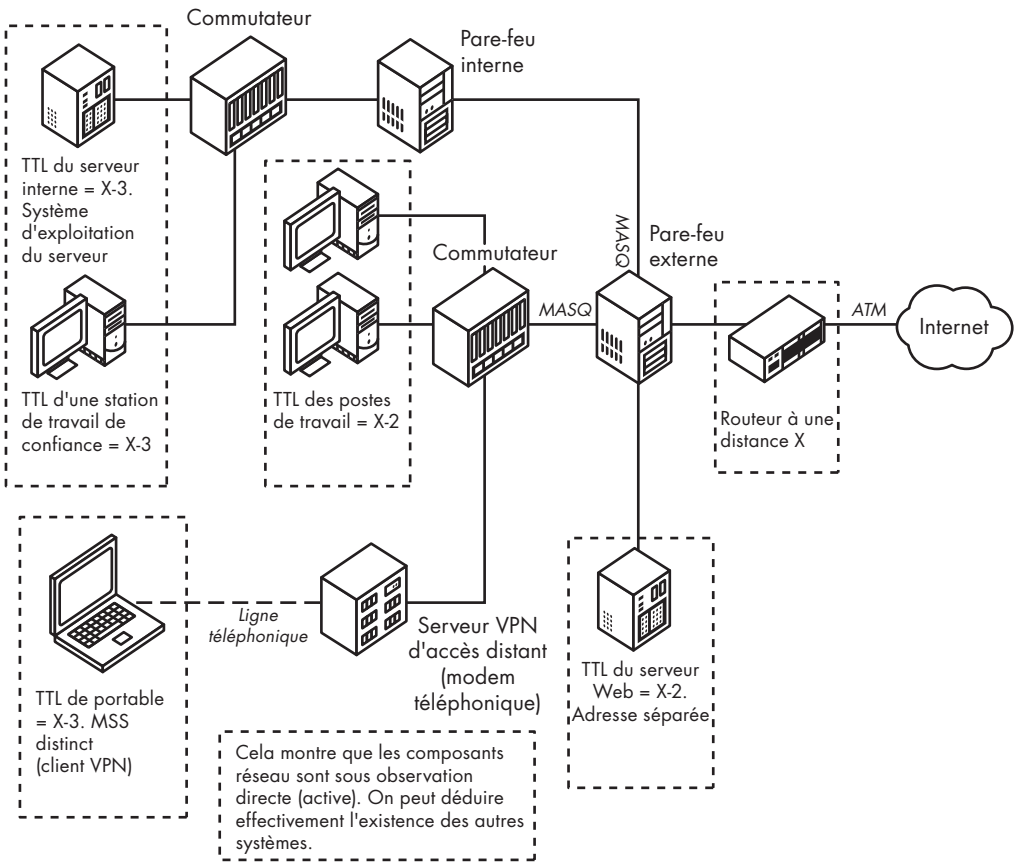

Commutateur
Pare-feu interne
TTL du serveur interne = X-3. Système d'exploitation du serveur
Commutateur
MASQ
Pare-feu externe
MASQ
ATM
Internet
TTL d'une station de travail de confiance = X-3
TTL des postes de travail = X-2
Routeur à une distance X
Ligne téléphonique
Serveur VPN d'accès distant (modem téléphonique)
TTL de portable = X-3. MSS distinct (client VPN)
TTL du serveur Web = X-2. Adresse séparée
Cela montre que les composants réseau sont sous observation directe (active). On peut déduire effectivement l'existence des autres systèmes.

Figure 10.1

On peut utiliser le fingerprinting passif pour établir la carte d'un réseau complexe et même inaccessible simplement en observant le trafic de certains de ces nœuds (le plus important étant de mesurer les caractéristiques du système d'exploitation ainsi que les valeurs TTL et MSS dans les paquets) et ensuite déduire la présence d'autres composants qui correspondent aux variations des caractéristiques observées. Je laisse au lecteur le soin de déterminer comment la cartographie de ce réseau pourrait être définitivement établie en observant simplement le trafic depuis l'extérieur.

les paramètres, une modification partielle peut être suffisante pour contrecarrer certaines tentatives d'analyse automatisées (hourrah !) ou pour tromper un responsable de la sécurité qui effectue une enquête sur un incident malveillant (oups !). Même si ce n'est pas un problème à grande échelle et que cela n'est donc pas une préoccupation pour l'analyse statistique, la question de la fiabilité peut susciter des inquiétudes dans des cas particuliers.

En outre, les capacités de suivi et de comptage des utilisateurs par fingerprinting que nous avons douloureusement disséquées au Chapitre 9 reposent presque entièrement sur la disponibilité de paramètres comme les informations de timestamp dans les paquets TCP/IP. Toutes les autres caractéristiques sont normalisées ou ont trop peu d'options possibles pour permettre d'identifier un seul ordinateur, sauf cas exceptionnel. Si de telles données ne sont pas disponibles parce que cette extension des performances spécifique a été désactivée (comme sur la plupart des systèmes Windows, par exemple), il n'est pas possible d'identifier un système avec précision.

Cela diminue la valeur potentielle des données à la fois pour les membres zélés d'une conspiration (qui, comme nous le savons tous, cherchent à connaître nos secrets les plus précieux), ainsi que pour les testeurs de sécurité ou les analystes des incidents (les experts en informatique judiciaire). Sans cette capacité d'identification fondée sur le timestamp, il peut être impossible de différencier plusieurs systèmes identiques fonctionnant derrière un masquerading ou d'identifier un individu dont l'IP a changé une fois qu'il a reconnecté son modem.

Cependant, une autre méthode de fingerprinting passif peut-être plus intéressante, prometteuse et exigeante comble facilement les lacunes du fingerprinting passif. Cette nouvelle approche rend extrêmement difficile d'induire en erreur un observateur distant et elle est presque universellement adaptée pour le suivi des systèmes. Encore plus intéressant, cette technique permet de distinguer plusieurs systèmes disposant exactement de la même configuration, en poussant la détection du masquerading à un tout autre niveau. Cette technique utilise les propriétés du mécanisme de génération du numéro de séquence au sein de TCP/IP et peut également donner lieu à quelques belles images.

Un bref historique des numéros de séquence

Comme nous l'avons vu au chapitre précédent, les numéros de séquence initiaux sont un mécanisme utilisé dans TCP pour assurer l'intégrité d'une session et donc garantir le niveau de sécurité le plus élémentaire.

La seule façon véritablement universelle de protéger une session TCP/IP non chiffrée contre l'injection de données, le détournement ou l'usurpation de la part d'un parfait inconnu est de s'assurer que les numéros de séquence sont sélectionnés d'une manière imprévisible pour l'attaquant. Cela réduit ses chances que sa supposition à l'aveugle soit correcte (et qu'un paquet qu'il usurpe soit accepté comme partie légitime de la session d'une autre personne), à tel point que ce risque est peu inquiétant dans le monde réel, même si l'attaquant lance une tempête réseau sur le système et envoie des milliers de paquets en espérant qu'au moins un d'entre eux aura un numéro de séquence à peu près correspondant.

Au début des années 1980, la sécurité des communications basées sur TCP ne semblait pas être un problème préoccupant : Internet était un environnement assez petit, autonome et

peut-être un peu élitiste utilisé par les scientifiques et leurs homologues. À ce titre, la spécification RFC du protocole TCP ne contenait aucune recommandation pour la sélection du numéro de séquence initial, si bien que presque toutes les premières implémentations de pile TCP/IP (et certaines implémentations ultérieures) utilisaient des algorithmes simples fondés sur le temps ou sur le compteur qui renvoyaient des suites de numéros pour les connexions. À l'époque, rendre ces numéros aléatoires apparaissait comme un gaspillage inutile de la précieuse puissance de calcul. En outre, cela ne pouvait qu'augmenter inutilement la probabilité d'une collision de numéros de séquence (une collision est une situation dans laquelle deux numéros ISN choisis pour se connecter à un hôte sont trop similaires, créant ainsi la possibilité que les anciens paquets arrivant d'une manière inopportune puissent être interprétés dans le contexte d'une mauvaise connexion. Naturellement, si l'on utilise des numéros aléatoires plutôt que des numéros croissant de façon séquentielle, il est plus probable à court terme que des collisions se produisent).

Bien entendu, Internet a énormément évolué depuis les années 1980, pour devenir rapidement plus accessible à un nombre grandissant d'utilisateurs différents. Au fur et à mesure que la quantité de données importantes circulant sur le réseau augmentait, les questions de sécurité devinrent plus pertinentes. Malheureusement, les mécanismes populaires et fiables de protection de la vie privée et de l'intégrité des données n'ont pas suivi l'expansion d'Internet : tous les services ne prennent pas en charge le cryptage, tous les utilisateurs ne savent pas comment l'utiliser et, plus important encore, la plupart des utilisateurs ne savent pas comment valider correctement les certificats cryptographiques fournis par les parties distantes.

Au fil du temps, et en particulier avec la généralisation des pratiques profitant de la faiblesse du mécanisme de génération ISN au milieu des années 1990 (même si ces abus se limitaient la plupart du temps aux services de discussion en ligne), il devint évident qu'une protection rudimentaire de l'intégrité des flux TCP/IP était nécessaire. Ceci était encore plus important pour la fraction marginale du trafic qui est effectivement protégé par cryptographie, car une interruption de la couche de transport causée par l'injection de données erronées ou de paquets RST est tout aussi néfaste, même si cela n'entraîne qu'une déconnexion (déni de service) et non l'injection de données.

Comme la seule manière de résoudre ce problème (sans une révision majeure de tout le système communication TCP connu ou presque) consistait à rendre le protocole difficile à attaquer en lui-même, de nombreux développeurs ont entrepris bien des efforts pour s'éloigner des mécanismes simples et dangereux de génération de numéros ISN s'incrémentant de un en un. Bien que ces efforts aient effectivement contribué à améliorer la résistance des connexions au blind spoofing, ils ont aussi ouvert plusieurs vecteurs intéressants de collecte d'informations qui permettent un fingerprinting des systèmes et des réseaux plus avancés, qu'il s'agisse d'évaluer la sécurité ou de planifier une attaque.

Obtenir plus d'informations des numéros de séquence

Naturellement, il est important de pouvoir distinguer les bonnes implémentations de générateurs ISN des mauvaises, aussi bien pour garantir la qualité que pour effectuer des tests de sécurité. Encore récemment, on évaluait généralement la qualité des numéros de séquence générés en analysant le code source ou en effectuant certains tests unidimensionnels sur le flux de bits des suites de numéros ISN, afin d'estimer l'entropie transportée par chaque bit en sortie. La première approche est souvent complexe et coûteuse, sujette aux erreurs et parfois impossible à réaliser (si le code source d'un système spécifique n'est pas mis à la disposition du public). La seconde méthode n'avait pas la capacité de capturer les dépendances plus subtiles de la séquence ni les autres caractéristiques d'un générateur d'une manière fiable et lisible, se concentrant plus sur les imperfections statistiques que sur la corrélation entre les valeurs retournées des connexion suivantes. De toute évidence, prouver qu'une implémentation est sécurisée en n'observant que sa sortie est à peu près impossible, mais il est facile de vérifier certains problèmes courants et de veiller à ce que l'algorithme sous-jacent soit raisonnablement robuste. Et, pourtant, même là, les méthodes que nous utilisions pour vérifier cela étaient au mieux plutôt faibles.

Aussi bien les conceptions de générateurs ISN non sécurisés originales que certains modèles d'aujourd'hui se fondent sur des systèmes arithmétiques itératifs et additifs qui calculent les nouvelles valeurs à partir de leur sortie précédente ; seuls la complexité de l'algorithme de calcul et le montant de l'imprévisibilité pratique introduits dans le processus semblent varier. Seules les conceptions sécurisées les plus récentes ne se basent pas sur l'arithmétique traditionnelle et utilisent des fonctions de création de résumé relativement lentes mais sécurisées par chiffrement pour implémenter des systèmes itératifs. Dans tous les cas, cependant, il serait intéressant de rechercher une corrélation complexe entre les résultats que le générateur produit pour les différentes connexions afin de détecter des failles éventuelles dans la conception de l'algorithme.

Il est évident que, si une dépendance apparente peut être observée entre la sortie du générateur ISN à un instant t et celle à un à instant $t+x$, l'attaquant peut choisir de se connecter avant la connexion qu'il espère brouiller ou usurper totalement, juste pour obtenir la sortie ISN à l'instant t. À partir de son observation du numéro de séquence renvoyé, il peut alors connaître la réponse qui sera ensuite générée par l'autre partie ($t+x$). L'attaquant peut donc usurper un paquet valide de cette nouvelle connexion même sans pouvoir observer directement l'ISN utilisé.

À partir de cette idée, j'ai effectué en 2001des recherches pour découvrir une méthode standard pour examiner les dépendances les moins évidentes des séquences ISN obtenues depuis des systèmes distants. Mon travail m'a amené à écrire un document dans lequel j'examinais certains des algorithmes de génération ISN plus en détail, en fournissant une méthode qui dépassait la détection des motifs et des défauts connus les plus évidents.

Dans ce document, intitulé "Strange Attractors and TCP/IP Sequence Number Analysis"[1], j'utilisais une méthode bien connue dans le monde des mathématiques appliquées, mais tout à fait nouvelle dans celui de la mise en réseau.

Coordonnées temporisées : images des séquences dans le temps

Lorsqu'il est question d'une boîte noire générant des ISN dans un des systèmes actuels dont les sources ne sont pas disponibles, on ne voit que sa sortie, une séquence de valeurs 32 bits véhiculées par les paquets TCP/IP et non l'algorithme. Pour de nombreux systèmes d'exploitation, ce code est propriétaire et bien gardé, hors de portée du simple mortel. Même dans un système open source, les sources peuvent être trompeuses et délicates, si bien qu'on peut reproduire les mêmes erreurs que le développeur original.

Le type d'entrée typique que nous aurions à évaluer ressemblerait probablement à ceci :

$S_0 = 244782$
$S_1 = 245581$
$S_2 = 246380$
$S_3 = 247176$
$S_4 = 247975$
$S_5 = 248771$
...

La logique de ces numéros est-elle immédiatement évidente ? Et, si c'est le cas, y a-t-il une méthode universelle que l'ordinateur suive pour créer ce motif et d'autres plus complexes ?

Une solution élégante semblait encore loin. J'espérais développer une méthode pour identifier certaines propriétés universelles de l'algorithme sous-jacent de l'ISN en observant uniquement sa sortie. Mais, avant de faire cela et afin de rendre l'analyse plus facile, il était souhaitable et tout à fait commode de supposer que, puisque de nombreuses implémentations sont fondées sur la répétition de certaines opérations arithmétiques, mieux valait observer les modifications entre les suites de résultats que leurs valeurs absolues. Regarder ces changements représente un avantage pour ces algorithmes et risque d'être utile également pour les autres générateurs possibles. Pour atteindre cet objectif, nous devons calculer un dérivé discret de la séquence d'entrée : les incréments entre les éléments de *S*. La séquence des deltas, *D*, commençant à l'évidence à $t = 1$, est donnée par l'équation suivante :

$$D_t = S_t - S_{t-1}$$

Dans cet exemple, la séquence de deltas qui en résulte est la suivante :

$D_1 = 799$
$D_2 = 796$

$D_3 = 799$

$D_4 = 796$

$D_5 = 799$

...

En négligeant les valeurs réelles et en ne regardant que la dynamique des numéros ISN, la dépendance sous-jacente devient plus évidente et le reste généralement pour toutes les implémentations qui s'appuient sur ce type d'arithmétique (pour les systèmes qui ne reposent pas sur une arithmétique itérative simple, cela n'a pratiquement aucun intérêt et n'aura pas d'influence significative sur la qualité des données aux fins de la présente analyse).

NOTE

Un chercheur particulièrement pédant compenserait également les irrégularités dans le temps au cours de l'acquisition de l'échantillon ; ici, nous supposons qu'une durée déterminée, une unité de base 1, survient toujours entre les acquisitions. Toutefois, pour une acquisition à haute vitesse, les performances du réseau et d'autres événements peuvent avoir de graves conséquences sur le temps. Pour garantir que ces différences dans le temps n'influencent pas les algorithmes qui utilisent l'entrée de l'horloge au cours du processus de génération de l'ISN, il serait plus sûr d'utiliser l'équation suivante (dans laquelle $T_t$indique le délai entre l'acquisition de S_{t_1} et de S_t) : $(S_t - S_{t\_1})/T_t$.

L'avantage de cette approche appliquée aux systèmes arithmétiques itératifs est assez évident. Les cas simples mis à part, cette méthode seule n'est cependant guère suffisante : on passe simplement d'une séquence de données sans relief assez difficiles à analyser à une autre.

Je décidai alors de convertir la séquence de deltas dans une forme qui puisse être facilement examinée par un homme ou par une machine pour les types de corrélations peut-être moins évidents que dans l'exemple précédent. Rien ne fonctionne mieux qu'un modèle en trois dimensions de la dynamique du système pour le premier groupe de données attendues. Malheureusement, les ISN nous fournissent juste assez d'informations pour tracer des images à une dimension, sur un seul axe. Alors, comment faire pour transformer ces informations en une forme tridimensionnelle claire ?

La solution est d'étendre l'ensemble des données en appliquant une stratégie de reconstruction coordonnée appelée *coordonnées temporisées*. Nous utilisons une méthode qui étend chaque échantillon en construisant des coordonnées virtuelles fondées sur les échantillons précédents dans la séquence. Si l'échantillon existant est considéré comme la valeur *x* de la coordonnée, on peut utiliser cette technique pour attribuer des valeurs *y* et *z* pour chaque échantillon existant et ainsi construire un triplet de coordonnées – *x*, *y* et *z* – suffisant pour placer chaque échantillon sur un seul point (ici, des pixels) dans un

espace tridimensionnel (cette technique n'est pas limitée à trois dimensions. Toutefois, pour visualiser et analyser des données, choisir un nombre plus élevé ne serait pas pratique. De toute façon, la plupart des êtres humains peinent à se représenter un plus grand nombre de dimensions, à moins d'être soûls).

Les coordonnées temporisées sont calculées de sorte que la deuxième coordonnée soit construite en utilisant la valeur échantillonné à *t-1*, que la troisième coordonnée corresponde à la valeur observée *t-2*, et ainsi de suite. Dans le cas présent, les coordonnées des données à l'instant *t* sont données par l'ensemble d'équations suivant :

$$x_t = D_t = S_t - S_{t-1}$$

$$y_t = D_{t-1} = S_{t-1} - S_{t-2}$$

$$z_t = D_{t-2} = S_{t-2} - S_{t-3}$$

Étant donné une séquence de triplets (*x*, *y*, *z*) pour un système dont on teste les dépendances dans le temps, il est possible de tracer le comportement d'un système de génération ISN dans un espace en trois dimensions. Comme l'emplacement d'un pixel représentant un échantillon donné dépend à la fois de la valeur "courante" et d'un certain nombre de résultats précédents, de nombreuses dépendances même assez complexes donnent comme résultat des motifs abstraits mais visibles montrant une densité irrégulière dans l'espace de la phase. On obtient ainsi un portrait unique de l'algorithme sous-jacent (quand il est utilisé en référence à ces portraits, le terme "*attracteur*"désigne une forme qui établit la cartographie de la dynamique d'un système. La forme (l'ensemble, l'espace) représente une "traînée" d'états à travers lesquels le système effectue des cycles ou évolue lorsqu'il est livré à lui-même).

La Figure 10.2 est une interprétation d'un ensemble de données qui, à l'origine, ressemblaient à ceci :

```
4293832719
3994503850
4294386178
134819 4294768138
191541
4294445483
4294608504
4288751770
88040492
...
```

Les Figures 10.3 à 10.5 illustrent plusieurs autres motifs de dépendance courants mais pourtant pas nécessairement évidents.

Figure 10.2

Une interprétation en trois dimensions de l'ensemble des données décrites dans le texte.

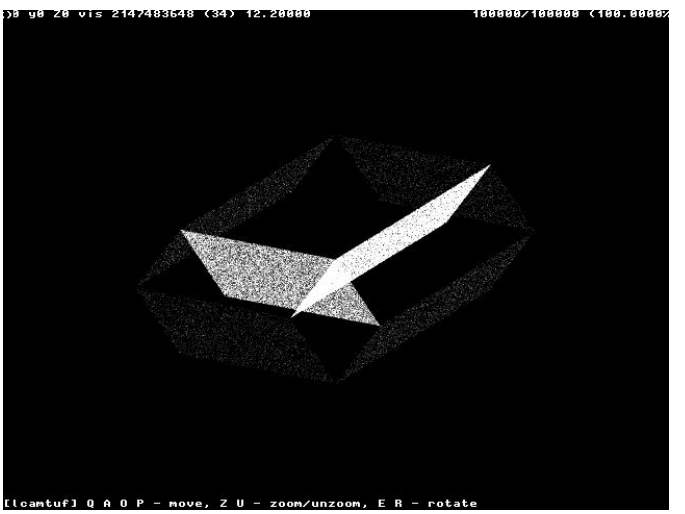


Figure 10.3

Une interprétation en trois dimensions d'un ensemble de données obtenues à partir d'une fonction de génération de nombres aléatoires complexe, mais non sécurisée.

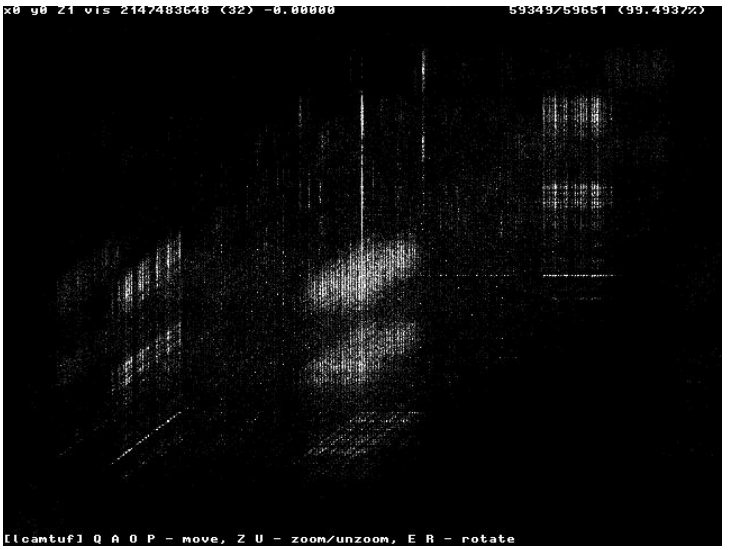


Figure 10.4

Interprétation d'un PRNG sans corrélation forte mais avec des tendances statistiques notables.

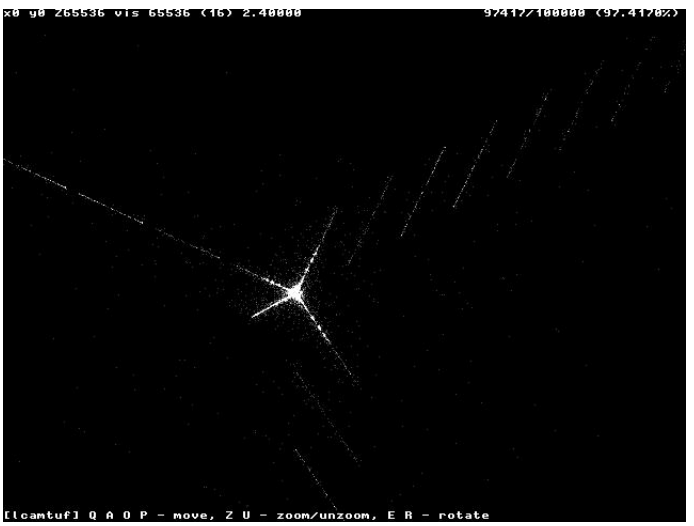


Figure 10.5

Un schéma courant de dépendance dans le temps, observé dans des conditions de test imparfaites.

Une galerie de jolies images de la pile TCP/IP

Cette méthode de visualisation semble parfaitement fonctionner et produire des motifs uniques, charmants ou souvent inquiétants pour de nombreuses implémentations que l'on croyait raisonnablement sécurisées. Beaucoup de ces images se trouvent disséminées dans les pages qui suivent. Mais ces images peuvent-elles nous apporter davantage qu'une représentation visuelle des paramètres et des caractéristiques difficiles à quantifier d'un générateur ? Un attaquant pourrait-il utiliser ces mystérieuses formes en trois dimensions de façon significative ? Un ordinateur pourrait-il les examiner d'une façon ou d'une autre et nous indiquer clairement ce qui convient ou non ? Est-ce qu'un générateur en forme de tournesol est plus facile à casser qu'un générateur en forme de brique ?

Avant de répondre à cette question, permettez-moi de faire une pause et d'incorporer quelques-unes des images les plus intéressantes que j'ai obtenues à l'époque où j'ai écrit le document original. Elles démontrent la grande diversité et la beauté de certains des motifs observés puisque, pour paraphraser l'ancien adage, une image en trois dimensions vaut mille mots. Les Figures 10.6 à 10.14 montrent les portraits des PRNG de plusieurs systèmes d'exploitation.

Toutes les images ne sont pas tracées à la même échelle et certaines formes sont nettement plus petites que d'autres. L'échelle et d'autres paramètres sont indiqués dans la première ligne de chaque image, comme le montre la Figure 10.6.

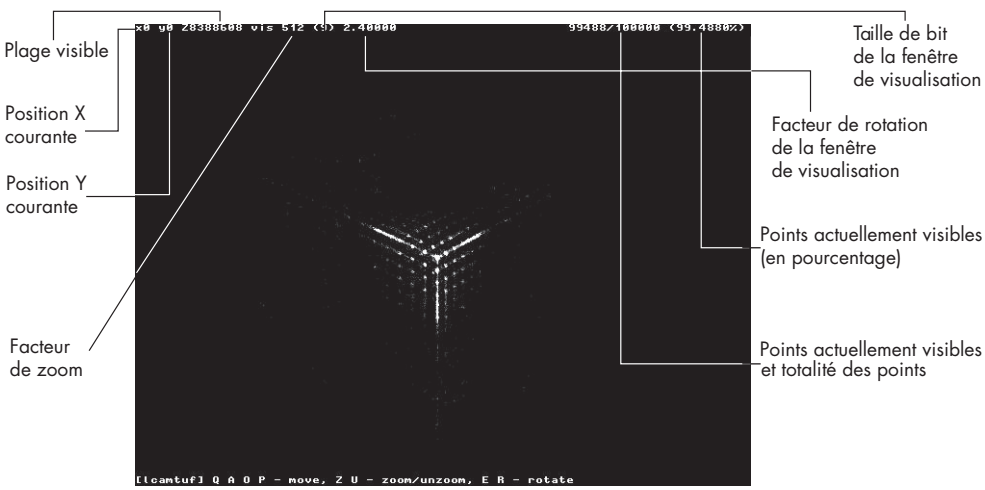

x0 y0 28388608 vis 512 (4) 2.40000
994887/100000 (99.4887%)
[loamtuf] Q A O P - move, Z U - zoom/unzoom, E R - rotate
Plage visible
Position X courante
Position Y courante
Facteur de zoom
Taille de bit de la fenêtre de visualisation
Facteur de rotation de la fenêtre de visualisation
Points actuellement visibles (en pourcentage)
Points actuellement visibles et totalité des points

Figure 10.6

Windows 98. L'ensemble illustré ici a un diamètre de 128 environ, ce qui indique que les prochains numéros ISN sont augmentés par un nombre transportant environ 7 bits de "hasard". Au sein de cet ensemble, la fréquence de répétition de certains points est élevée, comme dans l'un des exemples mentionnés dans la section précédente, ce qui indique peut-être une dépendance dans le temps pour tous les résultats. La taille de l'attracteur est petite, ce qui est inquiétant.

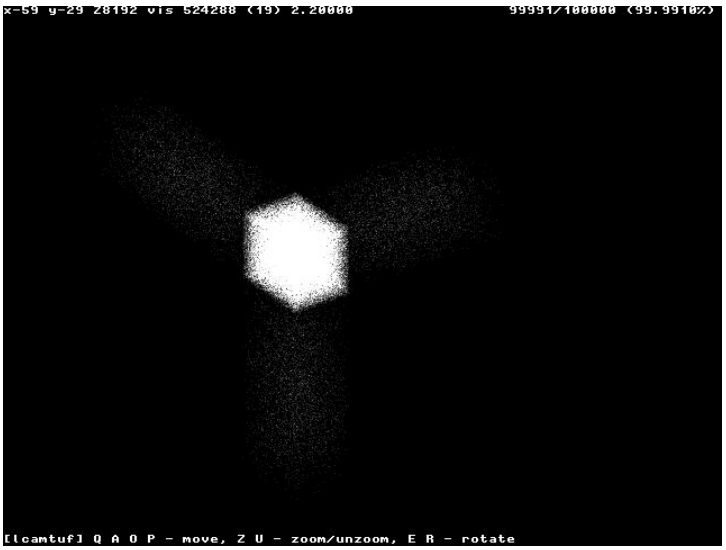


Figure 10.7

FreeBSD 4.2. Un cube uniforme d'une largeur de 16 bits, ce qui indique plutôt que l'incrémentation est faible mais réellement aléatoire à chaque étape.

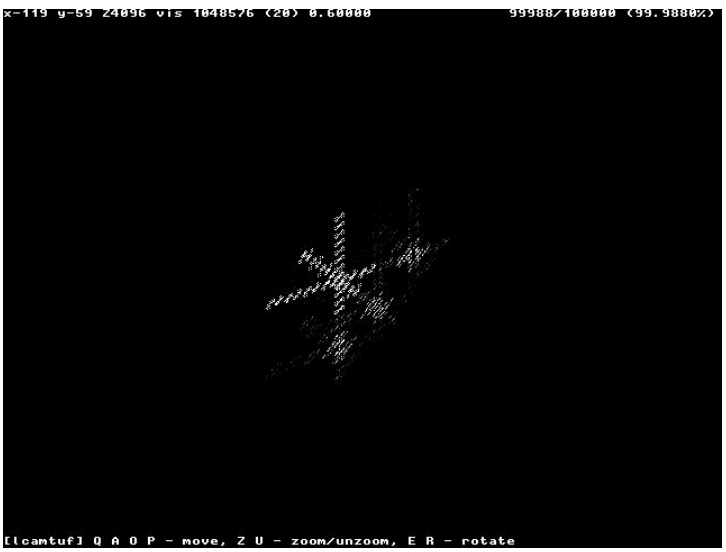


Figure 10.8

HP/UX 11. Une structure étrange en forme de X d'une largeur de 18 bits mais manifestement irrégulière. Plutôt un signe de niveaux de correspondance élevés d'un PRNG défectueux.

Figure 10.9

Mac OS 9. Une structure 17 bits semblable à la précédente mais légèrement différente.

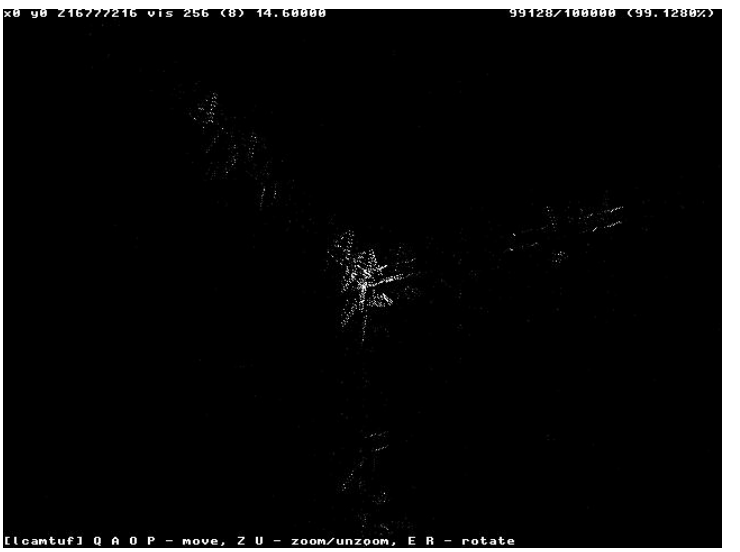


Figure 10.10

Windows NT 4.0 SP3. De nouveau, un motif d'attraction fort et un petit attracteur d'une largeur de 8 bits.

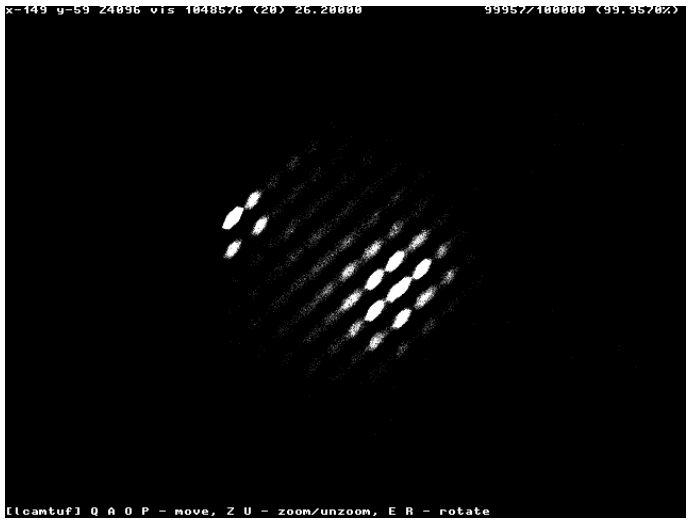


Figure 10.11

IRIX 6.5. Un nuage d'une largeur de 16 à 18 bits très irrégulier ; sans doute un algorithme défectueux.

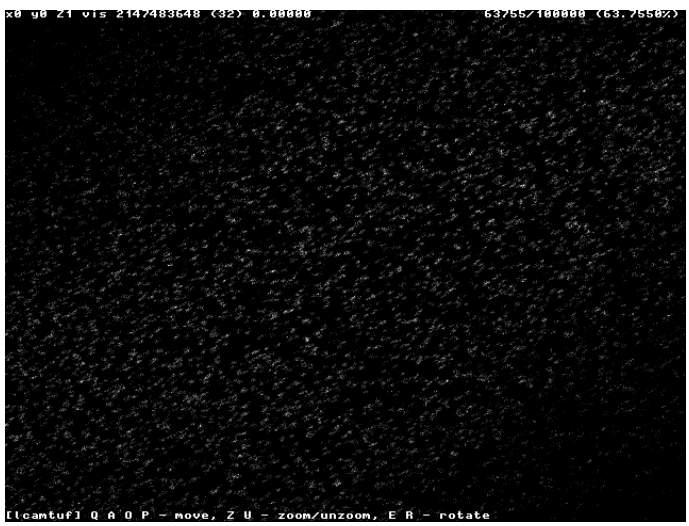


Figure 10.12

NetWare 6. Un système apparemment aléatoire. L'attracteur forme un nuage d'une largeur de 32 bits mais il est constitué d'un grand nombre de points répartis de façon très dense mais pas uniformément.

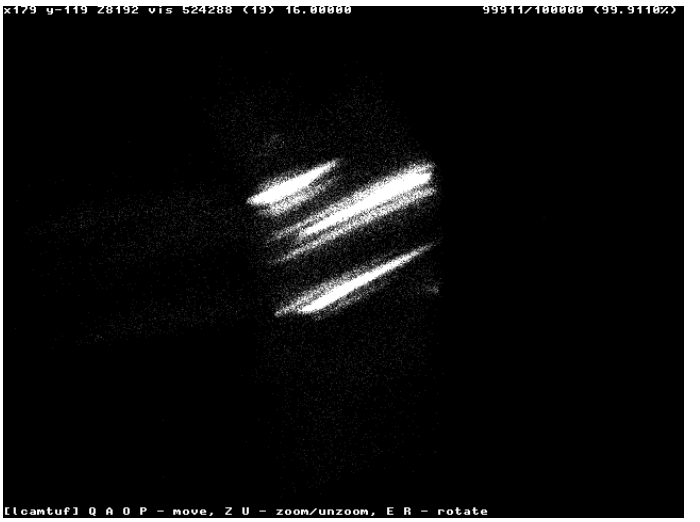


Figure 10.13

UNICOS 10.0.0.8. Un étrange nuage de 17 bits de largeur avec des bandes irrégulières de valeurs dont la probabilité est élevée.

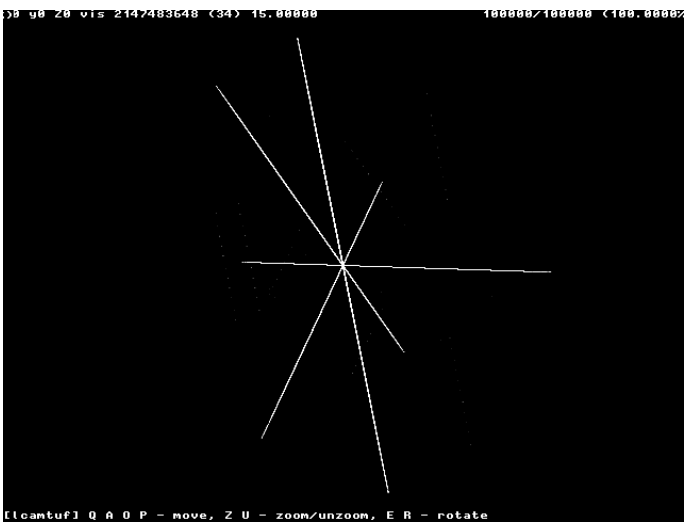


Figure 10.14

OpenVMS 7.2 (pile TCP/IP par défaut). Une structure d'une largeur de 32 bits avec une part faible de hasard, montrant des motifs de correspondance forts mais assez peu courants qui indiquent une conception de PRNG ratée.

Attaques utilisant les attracteurs

Revenons maintenant à la question du tournesol et de la brique. Oui, l'intérêt de ces belles images dépasse le simple plaisir visuel qu'éprouvent les geeks informatiques à les regarder. En fait, la structure de l'attracteur sur chaque système crée une matrice des motifs de comportements ISN possibles. La densité de l'attracteur correspond aux probabilités d'un type précis de dépendance dans le temps ou à des motifs statistiques dans le temps. Les régions de densité plus élevées dans l'attracteur correspondent aux corrélations qui se répètent le plus et qui sont aussi plus susceptibles de se produire à l'avenir tandis que les régions moins peuplées sont moins susceptibles d'être visitées. Et, donc, une fois l'attracteur d'un système spécifique approximativement tracé, l'attaquant peut deviner les résultats futurs. Mais comment ces formes permettent-elles d'établir une carte des valeurs ISN exactes ?

La clé d'une attaque réussie réside dans le fait de reconnaître que la coordonnée *x* de tous les points de l'attracteur dépend de la valeur de D_t, c'est-à-dire des numéros de séquence observés au temps *t* et *t–1* (car $D_t = S_t - S_{t-1}$). La coordonnée *y*, en revanche, dépend de D_{t-1} (ISN à *t–1* et *t–2*) et *z* dépend de D_{t-2} (ISN à *t–2* et *t–3*).

Supposons qu'un attaquant ait envoyé trois sondes à un système distant pour lequel la structure de l'attracteur du système d'exploitation a été cartographiée. Les sondes correspondent à des moments *t–3*, *t–2* et *t–1* et, évidemment, sont suffisantes pour reconstruire les coordonnées *y* et *z* du point qui indique le comportement de ce système à ce moment particulier sur la structure de l'attracteur.

L'attaquant peut utiliser cette information pour déduire les valeurs de *x* les plus susceptibles de se produire à partir des coordonnées *y* et *z* qu'il connaît, en fonction de l'observation des irrégularités dans la structure de l'attracteur qu'il a remarqué jusqu'ici. Les coordonnées *y* et *z* correspondent à une seule ligne perpendiculaire au plan *x* dans l'espace de l'attracteur (comme illustré à la Figure 10-15) – l'ensemble des points avec toutes les valeurs *x* possibles, mais aux coordonnées connues restantes. Les points situés sur la ligne d'intersection entre les zones de forte densité de l'attracteur ou près d'elles représentent les valeurs les plus susceptibles de correspondre à la coordonnée *x*. Les zones de plus faible densité sont, de toute évidence, moins susceptibles de correspondre à la valeur correcte de *x* car, après tout, ces points de l'attracteur ne sont pas apparus durant les mesures précédentes.

La possibilité d'établir une liste des valeurs *x* possibles une fois connues les coordonnées *y* et *z* représente une étape importante pour la réussite de l'attaque : en connaissant S_{t-1}(qui, vous vous en souvenez, a été précédemment obtenue par l'attaquant), on peut facilement calculer S_t pour chaque valeur *x* potentielle (D_t) de la façon suivante :

$$S_t = x + S_{t-1}$$

Ayant échantillonné trois numéros de séquence auparavant, S_{t-3}, S_{t-2}, et S_{t-1}, l'attaquant peut ainsi définir un ensemble de candidats probables pour le prochain numéro de séquence S_t que choisira le système attaqué pour la prochaine connexion – celle que l'attaquant n'a pas initiée mais qu'il espère brouiller. L'attaquant peut alors exécuter une attaque en envoyant des paquets TCP/IP contenant les numéros de séquence possibles ; il a besoin que ce numéro soit le bon dès le début, car toutes les suppositions erronées seront simplement ignorées par l'implémentation distante. Toutefois, dès que la valeur d'un des paquets falsifiés correspondra au numéro prévu dans la taille de fenêtre attendue, le trafic sera accepté, mettant ainsi en échec la protection de l'intégrité de la session offerte par TCP/IP.

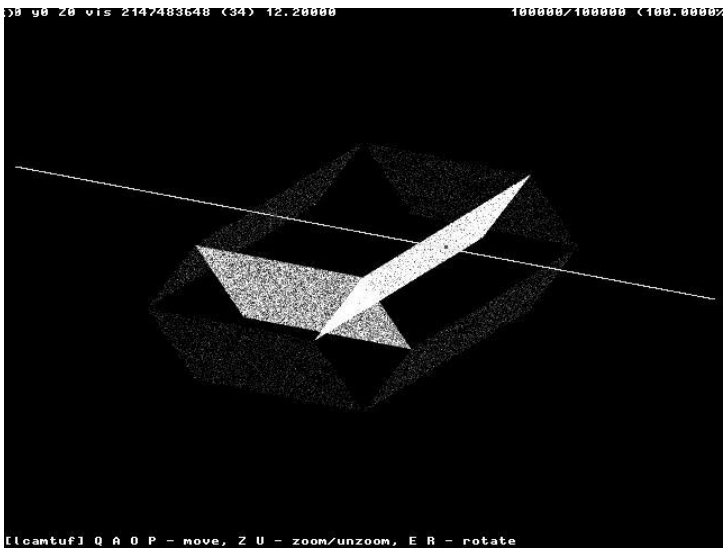


Figure 10.15

Une attaque "sur la ligne d'intersection" de l'attracteur.

Il faut bien sûr émettre quelques réserves sur cette attaque :

- La dynamique observée peut dépendre des conditions d'observation ou de la source elle-même. Cependant, à en juger par le taux de réussite obtenu en déployant cette technique contre des implémentations courantes, cela est peu probable.
- Si l'ensemble des candidats est particulièrement grand, comme avec les algorithmes qui produisent des nuages uniformes sans irrégularités claires, cette technique devient assez peu pratique car elle exige un trop grand nombre de tentatives pour faire une estimation correcte.
- Comme il est souvent peu pratique d'échantillonner la séquence entière des valeurs générées par une implémentation ISN dans un système (certains systèmes ont des

cycles longs, voire illimités), il est impossible de construire un attracteur complet. Pour parer à cela, il faut utiliser une approche par approximation : pour que la valeur soit choisie comme candidat, le point doit se trouver à l'intérieur d'un certain rayon depuis un point précis sur la ligne (*y*, *z*), ce qui compense le fait que même des zones assez denses de l'attracteur peuvent encore contenir des vides.

Pour que les résultats restent significatifs et établir une méthode d'évaluation comparative de la qualité d'un générateur ISN, j'ai décidé d'évaluer de façon empirique le taux de succès avec un nombre limité d'essais. Je voulais plus précisément déterminer la probabilité de tomber sur le bon numéro avec 5 000 tentatives, en me fondant sur l'hypothèse qu'un attaquant utilisant une connexion réseau à bas ou moyen débit ne pourrait envoyer qu'un maximum de 5 000 paquets dans un court laps de temps*.

Pour tester la validité de la démarche, j'ai choisi d'estimer la probabilité de succès en divisant les données d'entrée acquises depuis des systèmes distants en deux : une partie pour construire l'attracteur, et l'autre pour réaliser effectivement des tests. Le test lit simultanément quatre numéros de séquence qui se suivent et envoie trois d'entre eux à une implémentation qui doit ensuite générer un ensemble de 5 000 valeurs à partir des données de l'attracteur. Enfin, le résultat est comparé au quatrième numéro obtenu depuis l'ensemble des données de test. Ce test a été répété des centaines de fois pour obtenir des suites de quadruplets de numéros ISN pour chaque attracteur afin de déterminer un pourcentage de réussite approximatif, ce qui correspond dans la pratique aux chances qu'a l'attaquant de réussir grâce à cette approche.

Voici quelques-uns des résultats pour les systèmes de la galerie d'attracteurs.

Système d'exploitation	***Taux de réussite de l'attaque***
IRIX 6.5.15	25 % (25 tentatives sur 100)
OpenVMS 7.2	15 %
Windows NT 4.0 SP3	97 %
Windows 98	100 %
FreeBSD 4.2	1 %
HP/UX 11	100 %
Mac OS 9	89 %

* Le plus petit paquet SYN a 40 octets. Par conséquent, l'envoi de 5 000 paquets SYN consomme au moins 200 kilo-octets de bande passante. Cette quantité de données peut être transmise sur une ligne téléphonique avec un modem de compression des données V42.bis en 10 à 20 secondes. Le choix de ce seuil est tout à fait arbitraire mais semble raisonnable.

Cette approche se révèle assez efficace*, ce qui a incité de nombreux fournisseurs de logiciels à revoir leurs algorithmes ou les prétentions de leur algorithme de sécurité (la recherche suivante que j'ai publiée un an plus tard (2002) examinait certaines de ces modifications, et toutes n'étaient pas satisfaisantes).

Mais la véritable question est : qu'est-ce que cela a à voir avec le fingerprinting passif du système ?

Retour au fingerprinting du système

En effet, la capacité à cartographier la dynamique d'un générateur de numéro de séquence dans un système particulier et le fait que la plupart des implémentations présentent certains motifs dans l'analyse de phase plus ou moins uniques ont plusieurs conséquences vraiment fascinantes. La plus évidente est l'application du sondage de l'ISN à l'ancienne méthode de fingerprinting.

En observant plusieurs numéros de séquence obtenus à partir d'un système distant (par exemple lorsqu'une partie tente d'établir plusieurs connexions à un serveur), on peut essayer de trouver un attracteur qui corresponde le mieux à ces données en comparant l'échantillon observé avec une bibliothèque d'attracteurs connus (les chiffres n'ont pas besoin d'être prévisibles en utilisant la technique d'attaque décrite ici ; l'attracteur d'un système doit seulement pouvoir être distingué des autres).

Comparée au fingerprinting passif classique, cette méthode fournit généralement moins d'informations détaillées sur la configuration du système, mais elle est également presque infaillible. Pour déjouer cette technique, il faudrait modifier la manière dont les numéros de séquence sont générés. Mais il est le plus souvent impossible de régler de façon significative les paramètres de génération ISN depuis l'espace utilisateur, et modifier le noyau sans dégrader la sécurité demande généralement de solides connaissances et compétences (sans parler de l'accès aux sources).

Mais ce n'est bien sûr pas tout.

---

* Ces résultats s'appliquent à des scénarios dans lesquels une injection de données précises ou une usurpation est nécessaire. Si moins de précision est nécessaire ou si l'attaquant cherche uniquement à provoquer une rupture, la partie distante va accepter non seulement des paquets qui ont le numéro de séquence exact mais aussi ceux dont la taille de fenêtre correspond, comme spécifié dans les paramètres TCP/IP (voir Chapitre 9). En d'autres termes, les attaques par déni de service (DoS) seront encore plus réussies.

ISNProber, la mise en pratique de la théorie

En laissant de côté les images et la théorie, il serait bon de voir comment un échantillonnage ISN fonctionne en pratique et comment cela peut aider à évaluer la configuration d'un système distant ou à identifier les différentes machines qui utilisent cette configuration. Heureusement pour moi, il existe un programme qui mérite d'être mentionné.

Après avoir lu mon analyse de l'ISN TCP/IP, Tom Vandepoel écrivit un très bon utilitaire appelé ISNProber. ISNProber utilise l'analyse du numéro de séquence pour différencier plusieurs instances d'un même système, en se fondant sur le fait que deux systèmes distincts sont susceptibles d'être à différents endroits dans l'attracteur.

Pour simplifier, ISNProber peut dire que deux systèmes se cachent derrière une adresse partagée, en se basant sur l'apparence des numéros ISN observés. Par souci de clarification, supposons qu'un système Y utilise un générateur ISN qui incrémente le numéro d'un en un. À partir de l'adresse IP du site Web www.exemple.com, nous voulons déterminer combien il y a de systèmes. Nous identifions d'abord www.exemple.com comme étant le système Y, établissons ensuite plusieurs connexions, puis nous observons les numéros ISN qui suivent : 10, 11, 534, 13, 540 et 19.

Il semble évident que les numéros les moins élevés forment une séquence provenant d'un ordinateur qui gère moins de trafic ou a un uptime plus faible (10, 11, 13, 19), tandis que les chiffres les plus élevés correspondent à un autre système. Ainsi, deux ordinateurs partagent la même IP publique, peut-être derrière un répartiteur de charge. En outre, en variant les intervalles d'échantillonnage, nous pouvons examiner soigneusement le type de répartiteur de charge, sa politique de distribution des requêtes et le trafic qu'il reçoit.

Cette approche peut non seulement différencier les systèmes qui se cachent derrière une adresse commune mais également effectuer le suivi des utilisateurs du système Y lorsqu'ils passent d'une IP à une autre, tant qu'ils ne redémarrent pas leur machine (et donc remettent à zéro le compteur ISN). Pour les systèmes qui disposent de systèmes ISN plus sophistiqués que celui de notre exemple, cette distinction peut être plus difficile à réaliser, mais elle reste possible tant que les numéros ISN ne sont pas purement aléatoires sur l'ensemble des 32 bits (auquel cas des problèmes de collision se posent).

L'approche utilisée ici nécessite uniquement la présence d'une dose de prévisibilité dans l'algorithme de génération ISN. L'analyse de la séquence TCP/IP initiale semble donc représenter une alternative prometteuse ou un complément intéressant au fingerprinting traditionnel. Elle peut aussi malheureusement être très utile pour envahir la vie privée et effectuer le suivi des utilisateurs.

Empêcher l'analyse passive

Il est assez simple de se défendre contre les attaques par prédiction des numéros de séquence, et il existe depuis longtemps de bonnes solutions pour cela, comme la spécification RFC1948[2], de Steven M. Bellovin. Toutefois, empêcher l'analyse passive des chiffres est assez difficile, car ce problème non seulement relève de la faiblesse des algorithmes mais tient également au nombre d'algorithmes utilisés, car peu de systèmes partagent la même empreinte ISN. Même parmi les systèmes qui implémentent RFC1948 ou qui utilisent d'autres générateurs sécurisés par cryptographie et dont l'entropie est externe, les modes de comportement peuvent varier considérablement, selon les subtilités de l'algorithme et du choix des valeurs supposées suffire à contrecarrer une attaque que l'implémenteur a effectuée.

Un certain degré de sécurité peut être obtenu en déployant un pare-feu de paquets à états qui réécrit tous les numéros de séquence dans les paquets sortants*, rendant ainsi à peu près identiques tous les systèmes à l'intérieur d'un réseau protégé. Malheureusement, rares sont les systèmes à offrir cette fonctionnalité et rares sont ceux qui peuvent en bénéficier.

Matière à réflexion

L'utilité de la technique d'analyse de phase dépasse de loin la génération de numéro de séquence. Les autres paramètres qui sont choisis de façon pseudo-aléatoire ou selon certains schémas internes – comme les champs ID des paquets IP, les identificateurs de requête DNS (voir Figure 10.16), les cookies que l'application génère en "secret" pour identifier les sessions utilisateur, et ainsi de suite – peuvent être analysés avec succès, soit pour trouver des failles dans une conception soit pour identifier une implémentation et simplifier la poursuite de l'analyse ou faciliter une attaque.

---

* Solar Designer souligne que, techniquement, cela peut aussi être implémenté dans un pare-feu sans états sous la forme d'un hack. Le pare-feu peut combiner (par l'intermédiaire de XOR, par exemple) le numéro de séquence avec le hachage sécurisé d'une clé secrète et associer le tout à un quadruplet d'adresses et de ports qui identifient de façon univoque la connexion. Le hachage peut alors être enlevé des paquets en retour (là encore avec XOR), ce qui fait correspondre le paquet à l'idée qu'a l'hôte en interne de la connexion au moment de la livraison, bien que le paquet n'existe que dans une forme de 32 bits aléatoire et imprévisible lorsqu'il est à l'extérieur du pare-feu. Cela devrait fonctionner pour toutes les implémentations ISN, sauf les plus endommagées (qui répètent souvent les mêmes numéros et sont sujettes aux collisions).

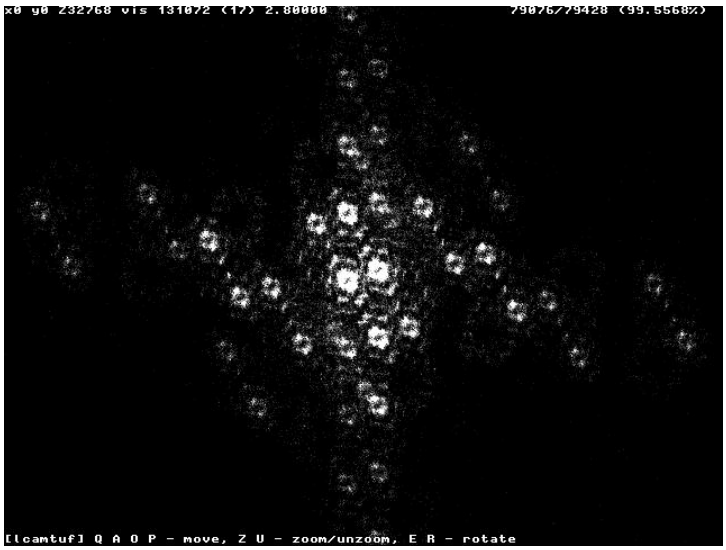


Figure 10.16

Un modèle d'attracteur intéressant pour l'implémentation du solveur de noms de Linux.

Des travaux dans ce sens ont été réalisés ou sont en cours. Dans un document en partie lié à mon étude originale, Joe Stewart offre un aperçu de certains des problèmes du système DNS[3] qui se posent en raison des avancées des mécanismes de prévision des numéros de séquence. Il constate que non seulement les méthodes de vérification des requêtes qu'offre le protocole DNS basé sur UDP sont insuffisantes pour résister à toute attaque par usurpation même, mais également que la faible qualité des identifiants de requêtes uniques générés par les différentes implémentations affaiblit encore ce modèle et le rend vulnérable à l'injection de données malveillantes. Étant donné que DNS est un des services fondamentaux d'Internet et qu'il est tentant d'usurper une réponse DNS d'un site populaire pour rediriger tous les utilisateurs d'un réseau en particulier sur une autre page Web, le *DNS poisoning* (empoisonnement du cache DNS) est une des menaces les plus sous-évaluées qui existent sur Internet.

Dan Kaminsky propose des images intéressantes et plus avancées des données prétendument aléatoires à l'adresse suivante : http://www.doxpara.com/pics/index.php?album=phentropy (voir Figure 10.17). Je vous conseille vraiment cette lecture.

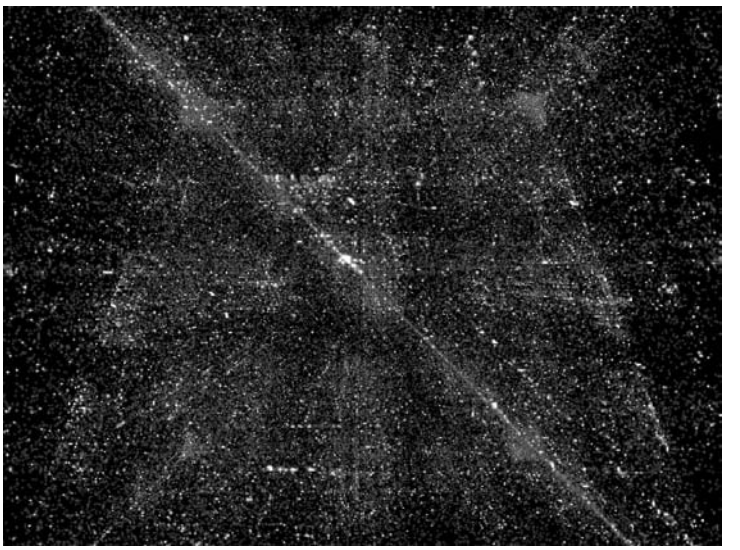


Figure 10.17

Rendu de l'aléa du noyau BSD effectué par Dan (reproduit avec l'accord de www.doxpara.com).

11

La reconnaissance des anomalies

Ou ce que les légères imperfections du trafic réseau peuvent nous apprendre.

Dans les chapitres précédents, j'ai disséqué et analysé un certain nombre de moyens pour extraire des morceaux d'informations potentiellement importantes à partir des paramètres "techniques" apparemment sans importance qui accompagnent tous les messages transmis par un suspect sur le réseau. Comme vous l'avez constaté, on peut obtenir une quantité considérable de données relatives à l'expéditeur sans que celui-ci ne sache qu'il les fournit (ou, s'il le sait, il n'apprécie guère d'être souvent incapable de ne plus fournir ces données). En utilisant bien des astuces pour analyser les flux et les paquets, on peut en théorie mesurer de nombreuses caractéristiques de la partie distante, connaître la signature d'un système en particulier ainsi que la configuration du réseau à partir de son comportement.

Cependant, la réalité est un peu différente : certains des paramètres observés ne correspondent pas vraiment à l'ensemble des valeurs normalement associées au périphérique spécifique ou à la configuration réseau que le suspect utilise. On peut bien sûr simplement ignorer ces divergences accidentelles, qui n'ont apparemment pas de sens, et toujours réussir à identifier l'origine du système ou effectuer le suivi de ses utilisateurs, mais il n'est pas forcément judicieux de le faire. On apprend à ne pas prêter attention à

ce genre d'ennuis dénués de sens, mais rien dans le monde de l'informatique ne se passe sans une bonne raison (du moins si l'on prend une définition assez large de "bonne"). En étudiant le mécanisme qui se cache derrière ces anomalies apparemment aléatoires et ces motifs minoritaires au lieu de les ignorer, on peut obtenir des informations précieuses sur des particularités de la configuration du réseau qui étaient invisibles jusque-là.

Dans ce chapitre, j'examine de plus près certains des processus qui peuvent affecter les caractéristiques observées d'un système. J'explique les raisons fondamentales, le but (ou l'absence de but), ainsi que les conséquences des technologies qui déclenchent un tel comportement.

Inutile de dire que la plupart des modifications reproductibles apportées aux paquets IP que j'examine ici proviennent de systèmes intermédiaires ayant une connaissance plus avancée de la topologie IP. Par conséquent, je vais commencer par un examen de deux sujets longtemps négligés : les pare-feu en général, et la traduction d'adresse réseau (NAT) en particulier.

Les pare-feu sont conçus pour rester des bastions solides, et moins on en sait sur ce qu'utilise l'autre mieux cela vaut pour lui. Pourtant, en dépit de politiques et de réglages rigoureux, les pare-feu gagnent en complexité et gèrent de mieux en mieux les problèmes de sécurité actuels, si bien qu'il devient de plus en plus facile de les examiner en utilisant des techniques de découverte du réseau indirectes ou passives.

Les bases des pare-feu de paquets

Les pare-feu populaires[1] sont, par essence, une catégorie de dispositifs de routage intermédiaires conçus pour aller à l'encontre de la conception originelle des périphériques de routage intermédiaires. Contrairement aux vrais routeurs, des systèmes qui sont censés prendre des décisions de routage non discriminatoires à partir des informations codées sur la troisième couche OSI, les pare-feu interprètent, agissent ou même modifient les informations sur les couches supérieures (la couche TCP, voire la couche HTTP). La technologie des pare-feu, bien que relativement récente, fournit un ensemble de solutions bien établies et bien comprises qui se retrouve aussi bien dans les réseaux à domicile que dans ceux des grandes entreprises. Les pare-feu sont configurés pour refuser ou autoriser certains types de trafics, les rediriger vers certains types de services et sont (ce qui n'est pas surprenant) utilisés pour restreindre l'accès à certaines fonctions et ressources pour l'ensemble du trafic qui les traverse. Ainsi, ils offrent une solution de gestion de réseau puissante, quoique parfois surestimée et sur laquelle on s'appuie trop.

La clé du succès des pare-feu dans tous les environnements réseau tient au fait qu'un seul composant relativement solide protège toute une gamme de systèmes complexes et fournit un dispositif de sécurité tolérant aux pannes lorsqu'un problème de configuration expose un service ou une fonction vulnérable sur un serveur protégé (dans les cas extrêmes,

les pare-feu sont simplement utilisés pour compenser la pauvreté des configurations et le manque de maintenance d'un système protégé, le plus souvent avec des résultats désastreux).

Filtrage et fragmentation sans états

Les pare-feu basiques sont des filtres sans états des paquets. Ils inspectent simplement certaines caractéristiques de chaque paquet, comme le port de destination pour les tentatives de connexion SYN sur le protocole TCP. Ils décident ensuite, uniquement à partir de ces caractéristiques, d'autoriser ou non le paquet à passer. La conception sans états utilise les ressources et la mémoire de façon extrêmement simple, fiable et efficace. Par exemple, un pare-feu sans états peut limiter les connexions entrantes d'un serveur de messagerie aux seuls messages adressés au port 25 (SMTP) et bloquer tous les paquets SYN sauf ceux qui s'adressent à ce port. Comme aucune connexion ne peut être établie sans ce premier paquet SYN, l'attaquant ne peut pas interagir de manière significative avec des applications sur d'autres ports. Pour obtenir ce résultat, le pare-feu n'a pas besoin d'être aussi rapide et complexe que le serveur de messagerie lui-même, car il n'a pas besoin de conserver une trace des connexions actuellement établies ni leur état exact.

Le problème avec ce type de protection aussi complètement transparente est que le pare-feu et le destinataire final peuvent comprendre certains paramètres différemment. Par exemple, imaginons qu'une personne malveillante parvienne à convaincre le pare-feu qu'il se connecte à un port autorisé mais conçoive son trafic de telle manière que le destinataire final le lise différemment et établisse une connexion à un port que le pare-feu est censé protéger. Un attaquant peut alors accéder à un service vulnérable ou à une interface d'administration et les ennuis commencent.

Bien qu'un tel malentendu semble improbable, il est assez facile à réaliser grâce à la fragmentation des paquets, en utilisant une approche communément appelée "attaque par recoupement de fragmentation"[2] (décrite en 1995 dans la spécification RFC1858). Dans cette situation, l'attaquant envoie un premier paquet contenant le début de la requête SYN à un port autorisé par le pare-feu de la victime (comme le port 25 déjà mentionné). Il ne manque au paquet qu'un petit bit à la fin et il dispose du flag "more fragments" configuré dans son en-tête IP, mais pourquoi le pare-feu devrait se soucier des données en fin de paquet ?

Le pare-feu examine le paquet et, comme il s'agit d'un paquet SYN, son port de destination est également examiné et jugé acceptable. Le paquet traverse le pare-feu, mais le destinataire ne l'interprète pas immédiatement (n'oubliez pas le processus de réassemblage traité au Chapitre 9). Au lieu de cela, le paquet est conservé et laisse en suspens la suite du réassemblage, ce qui ne se produira pas avant que le dernier segment de fin du paquet n'arrive.

Ensuite, l'attaquant envoie un second fragment de paquet. Ce deuxième paquet est créé pour recouvrir suffisamment le paquet initial et écraser le port de destination (l'un des champs de l'en-tête TCP) dans la mémoire tampon de réassemblage. Le fragment est conçu pour commencer à un décalage différent de zéro et ne contient pas la majeure partie de l'en-tête TCP, à l'exception du bit réécrit.

À cause de cela (et parce qu'il lui manque les informations nécessaires ou d'autres paramètres essentiels que le pare-feu utilise pour examiner les attributs d'un paquet TCP et déterminer s'il doit autoriser ou bloquer ce trafic), le deuxième fragment est relayé tel quel par un pare-feu sans états. Lorsque le destinataire le combine avec le premier paquet, ce deuxième paquet remplace le port de destination original par une valeur choisie par l'attaquant et ouvre effectivement une connexion sur un port qui devrait être protégé par le pare-feu.

Oups !

NOTE

Pour se protéger contre cette attaque, un pare-feu sans états bien conçu effectue une défragmentation initiale avant d'analyser les paquets. Toutefois, cela le rend un peu moins "sans états" et moins transparent.

Filtrage sans états et perte de synchronisation du trafic

Les filtres de paquets sans états ont aussi le défaut de ne pas être, et de loin, aussi restrictifs que nous pourrions l'espérer. Le filtrage ne peut être réalisé que si un seul paquet contient toutes les informations dont le filtre a besoin pour décider de la façon de le gérer. Or, après la poignée de main initiale, une connexion TCP est largement symétrique ; les deux parties ont des droits égaux et utilisent le même type de trafic (paquets ACK) pour échanger des données, si bien qu'il n'est pas facile d'appliquer des filtres ayant un sens en dehors de la phase initiale d'une connexion. Il n'existe aucun moyen de déterminer qui (si quelqu'un l'a fait) a initié la connexion à travers les paquets ACK qui sont échangés sans effectuer de suivi ni enregistrer les connexions. Ainsi, il est assez difficile de définir de façon significative quelle politique de filtrage le pare-feu doit tenter d'appliquer au transfert de paquets intermédiaires comme les paquets ACK, FIN ou RST.

Cette incapacité à filtrer le passage des paquets SYN n'est normalement pas un problème. Après tout, si un attaquant ne peut pas fournir les paquets SYN initiaux, il ne peut pas établir de connexion. Mais il y a un inconvénient : la façon dont les systèmes gèrent le trafic non SYN vers un port spécifique dépend si un port est fermé ou si le système est à l'écoute de ce port. Par exemple, certains systèmes d'exploitation utilisent

RST pour répondre aux paquets FIN errants et ne génèrent aucune réponse sur les ports qui sont dans un état ouvert (à l'écoute)*.

Des techniques comme le scan de paquet FIN ou ACK (cette dernière décrite à l'origine par Uriel Maimon[3] dans *Phrack Magazine*), ainsi que les scans NUL et Xmas (respectivement des scans à l'aide de paquets illégaux sans flags définis et avec tous les flags définis), peuvent donc être utilisées contre des filtres de paquets sans états pour découvrir quels ports sont ouverts sur un système distant ou pour établir quel trafic est rejeté par le pare-feu. Apprendre qu'un port spécifique est ouvert sans qu'il soit possible d'établir une liaison avec celui-ci ne constitue pas en soi-même une menace immédiate. Toutefois, un scan de cette nature révèle souvent des informations extrêmement précieuses sur les entrailles du réseau (le système d'exploitation et les services qui sont exécutés) qui peuvent ensuite être utilisées pour réaliser une attaque plus efficace et plus difficile à détecter une fois que la première ligne de défense est compromise ou contournée. Par conséquent, on considère ceci comme un point faible potentiel d'un pare-feu sans états.

La combinaison du mécanisme des cookies SYN et du filtrage sans états représente peut-être une menace plus grave. Les cookies SYN sont utilisés pour protéger les systèmes d'exploitation contre les attaques par épuisement des ressources, dans lesquelles l'attaquant envoie un très grand nombre de demandes de connexion falsifiées à l'hôte (ce qui n'est pas en soi une opération difficile à réaliser). Cela oblige le destinataire à envoyer des réponses SYN+ACK erronées et également à allouer de la mémoire et à consommer d'autres ressources pour ajouter cette connexion potentielle à ses tables d'état TCP. La plupart des systèmes qui subissent une telle attaque consomment trop de ressources et ralentissent énormément ou refusent à un moment de servir tous les clients, jusqu'à ce que ces fausses connexions se terminent.

Pour faire face à ce problème potentiel, les cookies SYN utilisent une signature chiffrée (un résumé, en fait, qui identifie la connexion sans équivoque) dans toutes les réponses SYN+ACK à l'intérieur du champ ISN, puis oublient complètement la connexion. La connexion n'est ajoutée à la table d'état qu'après l'arrivée de la réponse ACK de l'hôte, et uniquement si le numéro d'acquittement est validé par la procédure de chiffrement.

Cependant, le problème avec les cookies SYN dans une telle conception vient du fait qu'il est possible que le paquet SYN (et la réponse SYN+ACK) n'ait jamais été envoyé en premier lieu. Si l'attaquant peut créer un cookie ISN validé par l'algorithme de cookie SYN de l'hôte (peut-être parce que l'attaquant dispose de suffisamment de bande passante ou parce que l'algorithme est faible), il peut envoyer un paquet ACK qui poussera l'hôte distant à ajouter une nouvelle connexion à sa table d'état même s'il n'envoie

---

* Certains aspects de ce comportement (la tendance à répondre par RST aux paquets errants et inattendus envoyés aux ports fermés et à rejeter le même trafic adressé aux ports sur lesquels un service écoute les connexions) sont définis par RFC 793, tandis que d'autres relèvent uniquement d'un choix effectué lors de l'implémentation.

jamais de paquets SYN ni ne reçoit de paquets SYN+ACK. Un pare-feu sans états n'aura aucun moyen de savoir qu'une connexion vient d'être établie, car il n'a jamais reçu la demande d'ouverture ! Comme il n'existe pas de paquet SYN initial, le pare-feu n'a pas pu vérifier l'adresse IP et le port de destination et donc ni les approuver ni les rejeter et, pourtant, une connexion est tout d'un coup établie.

C'est vraiment mauvais.

Les filtres de paquets à états

Pour résoudre les problèmes des filtres sans états, il faut stocker sur le pare-feu certaines informations sur le trafic précédent et l'état des flux établis. C'est la seule façon de prévoir de façon transparente les résultats de la défragmentation ou d'obtenir le contexte des paquets en cours de transfert et de décider s'ils sont illégitimes et doivent être rejetés ou s'ils sont attendus par le destinataire et doivent être livrés.

Les ordinateurs hautes performances devenant plus abordables, il est devenu possible de concevoir des systèmes de pare-feu beaucoup plus complexes et plus avancés que nous ne pouvions l'imaginer auparavant. Ainsi, nous en sommes arrivés au suivi des états de la connexion, une situation dans laquelle le pare-feu non seulement examine chaque paquet mais se souvient également du contexte d'une connexion et valide chaque paquet en fonction de ces données. Cela permet au pare-feu de surveiller étroitement le réseau et de supprimer tout trafic indésirable ou inattendu sans compter sur la capacité du destinataire à distinguer ce qui est légitime de ce qui ne l'est pas. Les filtres de paquets à états essaient de suivre la piste des connexions et n'autorisent que le trafic appartenant à l'une des sessions actives. En conséquence, ils améliorent la protection et les capacités de journalisation.

La tâche des filtres à états est bien entendu plus difficile que celle des filtres sans états et consomme beaucoup plus de ressources, surtout si ce dispositif protège un réseau de grande taille. Dans ce cas, le pare-feu nécessite beaucoup de mémoire et un processeur rapide pour stocker et surveiller ce qui se passe sur le câble.

Une analyse à états est également plus susceptible d'entraîner des problèmes ou de créer une certaine confusion. Les ennuis surviennent dès que le pare-feu et les terminaux comprennent différemment l'état actuel d'une session TCP/IP, ce qui n'est pas improbable compte tenu de l'ambiguïté des spécifications et le nombre de piles différentes utilisées. Par exemple, lors de la réception d'un paquet RST qui n'entre pas dans les limites des numéros de séquence acceptés par le destinataire, un pare-feu qui inspecte les numéros de séquence moins rigoureusement que le destinataire final peut conclure que la connexion est terminée, tandis que le destinataire pense, lui, que la session est encore ouverte et s'attend à recevoir d'autres communications de cette connexion, ou *vice versa*. En fin de compte, l'inspection à états a un prix.

Réécriture de paquet et NAT

Pour améliorer l'interprétation des paquets et assurer une meilleure protection contre les attaques, notamment celles qui utilisent la fragmentation des paquets pour contourner les règles du pare-feu, on a donné la possibilité aux pare-feu de relayer mais aussi de réécrire des portions du trafic transmis. Par exemple, une approche consiste à résoudre l'ambiguïté en effectuant une défragmentation (réassemblage) obligatoire du paquet avant de confronter le paquet aux règles d'accès configurées par l'administrateur réseau.

Avec le développement de solutions plus sophistiquées, il est devenu évident que la réécriture des paquets non seulement bénéficierait au réseau mais constituerait également un pas de géant pour la sécurité et la fonctionnalité du réseau, notamment en déployant des technologies très utiles comme NAT. NAT est un mécanisme qui permet de faire correspondre certaines adresses IP à un ensemble différent d'adresses IP avant de les transmettre et de reconstituer les réponses renvoyées par un système protégé. Un mécanisme NAT à états peut entre autres être utilisé pour implémenter des configurations tolérantes aux pannes dans lesquelles une seule adresse IP publique est desservie par plus d'un serveur interne. Pour économiser l'espace d'adressage et améliorer la sécurité des communications sur Internet, NAT peut être implémenté pour permettre aux hôtes d'un réseau interne de "masquer" leurs adresses privées derrière une seule adresse IP publique.

Dans le premier scénario, NAT réécrit les adresses de destination sur les paquets entrants vers un certain nombre de systèmes privés situés derrière le pare-feu. Cette configuration tolérante aux pannes répartit la charge de façon équilibrée entre plusieurs systèmes pour les requêtes effectuées vers un site Web populaire (http://www.microsoft.com, par exemple) ou d'autres services essentiels. Si un système échoue, d'autres systèmes peuvent prendre la relève. Cette tâche est parfois réalisée par des périphériques dédiés (appelés répartiteurs de charge), mais elle est souvent également prise en charge par des pare-feu sur lesquels le NAT est activé.

Le deuxième scénario, communément appelé masquerading, s'appuie sur la réécriture des adresses sources sur les paquets sortants de sorte qu'un certain nombre de systèmes privés et protégés (qui peuvent utiliser des adresses privées non routées vers ce réseau depuis Internet, comme 10.0.0.0) peuvent se connecter au monde extérieur, puisque leurs connexions sortantes sont interceptées et réécrites par le pare-feu. Les systèmes sont cachés derrière un pare-feu et, pour les destinataires situés en dehors du réseau protégé par NAT, leurs actions semblent provenir du pare-feu. La connexion est transférée à une adresse IP publique et à un port spécifique, puis le trafic est envoyé vers l'extérieur. Tout le trafic en retour qui arrive à cette adresse IP et ce port est réécrit puis relayé au système privé qui a initié la connexion et transmis au réseau interne. Cela permet à l'ensemble des stations de travail du réseau privé qui ne sont pas censées offrir de services

Internet de ne pas être directement accessibles depuis le monde extérieur, ce qui accroît considérablement la sécurité du réseau, dissimule une partie de sa structure et préserve l'espace d'adressage IP public, qui, autrement, aurait du être acheté pour tous les systèmes. Grâce à ce système, une partie qui ne possède qu'une seule adresse IP publique peut créer un réseau composé de centaines ou de milliers d'ordinateurs qui disposent tous d'un accès à Internet.

Lost in Translation

Une fois de plus, la traduction des adresses est plus complexe qu'elle ne semble l'être : certains protocoles de plus haut niveau ne permettent pas aussi simplement d'établir une connexion à un système distant et d'envoyer un groupe de commandes. Par exemple, l'ancien mais très populaire protocole FTP[4] (*File Transfer Protocol*) repose dans son mode le plus simple et le plus largement pris en charge sur la création d'une connexion en retour (direction inverse) depuis le serveur vers le client pour transférer les données demandées ; la connexion initiale initiée par le client est utilisée uniquement pour émettre des commandes. Beaucoup d'autres protocoles – notamment certains protocoles de discussion, les utilitaires de collaboration P2P ou de partage de données, les services de diffusion multimédias – utilisent aussi des conceptions étranges ou inhabituelles qui font appel aux connexions inverses et au saut de port ou qui autorisent un trafic en particulier à revenir à la station de travail, même s'il ne conserve pas de "contexte serveur" pour la session de l'utilisateur (comme les paquets UDP [*User Datagram Protocol*]).

Pour relever ces défis, chaque implémentation du masquerading qui ne vise pas à rendre ces protocoles inutiles doit être équipée d'un certain nombre d'assistants de protocole. Ces protocoles inspectent les données de l'application échangées au sein d'une connexion, réécrivent parfois une partie de celles-ci et ouvrent temporairement des trous dans le pare-feu pour permettre une connexion en retour.

D'où un autre problème, identifié pour la première fois dans l'assistant FTP par Mikael Olsson il y a plusieurs années[5] et étudié ensuite dans d'autres assistants de protocole par l'auteur de ce livre, entre autres[6]. Le problème est que ces assistants décident d'ouvrir des trous dans le pare-feu en se fondant sur les informations transmises par une station de travail au système distant sur un protocole spécifique. Ils supposent que le trafic généré par le système est transmis à l'utilisateur au nom et à la connaissance de l'utilisateur. Inutile de dire que certains programmes, comme les navigateurs Web, peuvent être abusés et envoyer certains types de trafics réseau, y compris du trafic qui "ressemble à" un protocole que le programme ne prend pas en charge nativement. Ils peuvent même être contraints de le faire automatiquement en créant du contenu malveillant spécifique et en l'envoyant à l'application. Ce trafic falsifié peut tromper un assistant du programme à ouvrir un trou dans le pare-feu.

Un exemple classique de cette attaque consiste à abuser un navigateur Web générique : en ajoutant une référence à une page Web ou à un élément Web prétendument situé sur un port HTTP non standard du système d'un attaquant (ce qui est cependant tout à fait standard pour le trafic FTP), le client peut être contraint de se connecter à cette ressource et essayer d'émettre une requête HTTP. Comme le port sur lequel la connexion est établie est normalement utilisé par FTP, l'assistant FTP du pare-feu commence l'écoute de la conversation pour apporter son aide si cela se révèle nécessaire.

Cet exemple de lien URL force le client HTTP à se connecter au port FTP et à émettre ce qui semble être la commande FTP PORT, qui sera reprise par l'assistant du pare-feu :

```
HTTP://SERVER:21/FOO<RETURN>PORT MY_IP,122,105<RETURN>
```

La requête émise par le client serait juste un charabia dénué de sens pour un service FTP légitime sur l'autre système, et la réponse du service serait incompréhensible pour le client Web émettant cette requête, mais là n'est pas la question. Ce qui importe, c'est que l'attaquant peut contrôler une partie de la requête – le nom du fichier que le client demande au serveur. Ce nom de fichier fictif, choisi par l'attaquant, peut contenir toutes les données que l'attaquant désire. En intégrant au nom de fichier des sous-chaînes normalement identifiées comme des requêtes FTP, l'attaquant peut amener un assistant de protocole FTP qui écoute cette connexion dans l'attente d'une commande textuelle en particulier (PORT) à croire que l'utilisateur tente de télécharger un fichier spécifique. Ainsi, le serveur distant est temporairement autorisé à se connecter à la victime (ici, à un port 31337 dont le numéro semble malveillant – 122*256+105=31337). La victime laisse donc entrer l'attaquant sans le savoir. Encore une fois, ce n'était pas ce qui était prévu...

Les conséquences du masquerading

Tous les scénarios mentionnés précédemment sont liés à l'abus du masquerading, mais la simple présence du masquerading lui-même peut fournir des informations intéressantes sur une autre partie.

Comme indiqué précédemment, le masquerading est intrusif. Son principe opératoire de base consiste à modifier le trafic sortant en réécrivant certaines portions de celui-ci. Ce faisant, il ne se contente pas de modifier simplement l'adresse, ce qui rend non seulement possible de le détecter mais permet aussi à un observateur attentif d'identifier le système particulier de pare-feu qui est utilisé. Plus précisément, lorsque le masquerading est utilisé, on peut rencontrer certains des changements suivants :

- On peut observer un écart entre le TTL des paquets qui arrivent et la distance mesurée ou prévue du réseau de destination. Le trafic qui est émis par un réseau utilisant le masquerading a au moins un saut "plus ancien" qu'un paquet provenant d'un

système dont l'adresse IP pour les connexions sortantes provient directement d'un réseau protégé.

- Dans la plupart des cas, plusieurs systèmes d'exploitation ou différentes configurations système (ou uptime) peuvent être trouvés dans le réseau d'origine. Ces systèmes ont des caractéristiques TCP/IP légèrement différentes, comme nous l'avons vu aux Chapitres 9 et 10. En observant différentes empreintes TCP/IP dans les connexions provenant apparemment de la même adresse IP, on peut savoir si le NAT est présent sur une machine et qu'il existe un réseau interne derrière elle.
- Enfin, un observateur distant est susceptible de noter une *modification du port source*. Il s'agit d'un autre incident inhabituel qui survient car les connexions provenant du réseau utilisent des ports source éphémères qui ne font pas partie de ceux que le système d'exploitation utilise normalement.

Chaque système d'exploitation réserve une plage particulière de ports source pour créer un identifiant local de toutes les connexions sortantes. Toutefois, un pare-feu utilise souvent une autre série de ports spécifique au périphérique NAT du système d'exploitation pour établir la cartographie des connexions utilisant le masquerading. Dans ce cas, si on observe une gamme de ports différente de celle attendue pour le système d'exploitation détecté (si par exemple Linux, qui fonctionne normalement sur les ports 1024 à 4999, semble utiliser des numéros de port très élevés), il est possible de déduire la présence d'une traduction d'adresse et parfois même de déterminer le type de pare-feu utilisé.

Ces techniques de base sont couramment utilisées pour détecter le masquerading et identifier les réseaux qui utilisent le masquerading. Mais plusieurs autres moyens de détecter la réécriture des paquets sont également disponibles.

La taille des segments

Un moyen moins évident et donc moins courant de détecter les dispositifs de réécriture des paquets et d'en apprendre plus sur la configuration du réseau consiste à analyser la taille maximale d'un segment dans le trafic entrant.

Comme la fragmentation des paquets IP ajoute une surcharge notable au trafic fragmenté, il est souvent considéré comme nuisible aux performances, si bien que de nombreux développeurs essaient de l'empêcher lors de l'implémentation. Mais, comme nous l'avons vu plus tôt, la fragmentation est difficile à éliminer puisqu'il semble à peu près impossible de déterminer avec précision, rapidement et de façon fiable l'unité de transmission maximale (MTU) sur un chemin avant la communication. Même la meilleure méthode disponible, PMTU, est loin d'être parfaite et a toujours un impact sur les performances lorsqu'elle est déclenchée. Pour lui permettre de détecter par

tâtonnements le bon paramètre MTU, il peut être nécessaire d'abandonner et de renvoyer quelques paquets qui ne conviennent pas.

Pour éviter l'impact du PMTU sur les performances et la fiabilité et afin de réduire la charge de la fragmentation, de nombreux pare-feu NAT qui réécrivent certains paramètres du trafic sortant modifient également le paramètre MSS (la longueur maximale du segment) dans les en-têtes TCP des connexions provenant du réseau privé au profit d'un MSS plus approprié pour le lien externe du réseau. Ce nouveau paramètre est généralement plus court (il a un MTU moins élevé) que celui du réseau LAN. Cette modification garantit que le destinataire ne tente pas d'envoyer des données qui ne conviendraient pas à la liaison si celle-ci traverse la partie de l'infrastructure qui a le MTU le plus faible, rendant ainsi la fragmentation moins probable (cela suppose que toute incompatibilité MTU a plus de chances de se produire près du système de l'expéditeur ou du destinataire, là où se trouvent généralement les différents types de liaisons où le MTU est le plus faible, comme les connexions DSL ou les réseaux sans fil, et où la taille des paquets doit être "réduite" pour passer).

Cette réduction du paramètre MSS n'est pas particulièrement facile à détecter. En fait, il est impossible de dire si le MSS a été fixé à une valeur donnée par l'expéditeur ou modifié quelque part en chemin. Du moins à une exception près. Souvenez-vous du Chapitre 9 : l'algorithme de sélection de la taille de fenêtre a quelque chose de spécial sur de nombreux systèmes actuels.

Le paramètre Taille de fenêtre détermine la quantité de données qui peut être envoyée sans acquittement. Le développeur définit souvent son paramétrage en fonction de certaines règles ou croyances qui relèvent plus du mysticisme qu'autre chose. Dans les deux approches les plus populaires, la valeur doit être un multiple du MTU moins les entêtes de protocole (une valeur appelée Longueur maximale du segment, ou MSS) ou être tout simplement un chiffre suffisamment élevé et "rond". Les anciennes versions de Linux (2.0) utilisaient comme valeurs des puissances de 2 (16 384, par exemple). Dans Linux 2.2, cette valeur était un multiple du MSS (onze ou vingt-deux fois le MSS, pour une raison quelconque), tandis que les versions les plus récentes de Linux utilisent une valeur égale à deux ou quatre fois le MSS. La Dreamcast, une console permettant d'accéder au réseau, utilise une valeur de 4 096 et Windows utilise souvent 64 512.

Un nombre sans cesse croissant de systèmes actuels (y compris les nouvelles versions de Linux et de Solaris, certaines versions de Windows, et SCO UnixWare) utilisent une taille de fenêtre qui est un multiple du MSS. Ainsi, il est facile de dire quand le réglage du MSS dans un paquet a été altéré, car la taille de la fenêtre du paquet n'est alors plus un multiple du MSS. En fait, il est probable qu'elle ne puisse plus être divisée du tout par le MSS.

En comparant le MSS à la taille de la fenêtre, on peut détecter de façon fiable la présence d'un groupe de pare-feu qui prennent en charge le *MSS clamping* (son réajustement pour

qu'il corresponde à la liaison) sur de nombreux systèmes. Bien que le MSS clamping soit facultatif sur Linux et FreeBSD, il est souvent effectué automatiquement sur les pare-feu et les routeurs DSL des réseaux domestiques. Ainsi, la présence d'un paramètre MSS anormal indique non seulement la présence d'un périphérique de réécriture des paquets, mais aussi une capacité NAT associée, ce qui peut être considéré comme un indicateur de la connexion réseau de l'expéditeur.

Suivi à états et réponses inattendues

Le suivi d'une connexion à états et la réécriture des paquets ont une autre conséquence importante : certaines réponses exigées par les RFC sont générées par le pare-feu, et non par l'expéditeur. Cela permet à un attaquant de détecter et de sonder un tel périphérique assez efficacement. Lorsque la connexion est effacée depuis la table d'états NAT (en raison de son expiration ou si un des terminaux envoie un paquet RST de fin de connexion qui n'atteint pas l'autre terminal), le trafic de cette session n'est alors plus transmis au destinataire, comme cela serait le cas avec des filtres de paquets sans états, mais est géré directement par le pare-feu.

Les spécifications TCP/IP exigent qu'un destinataire réponde à tous les paquets ACK imprévus par RST, afin d'informer l'expéditeur que la session qu'ils tentent de prolonger n'est plus honorée par le destinataire ou ne l'a jamais été. Certains pare-feu peuvent violer ces spécifications RFC et refuser de répondre à ce trafic, en supprimant les paquets qui ne semblent pas appartenir à une session existante (ce qui n'est pas toujours sage, car cela peut causer des retards inutiles lorsqu'une connexion légitime est abandonnée en raison de problèmes intermittents sur le réseau).

De nombreux périphériques répondent toutefois avec un paquet RST légitime et attendu. Cela ouvre une nouvelle voie pour la détection et le fingerprinting du pare-feu. Comme le paquet est créé de toutes pièces par le pare-feu, ses paramètres concernent le pare-feu et non pas ce que le pare-feu protège. Cela permet d'utiliser les techniques de fingerprinting habituelles mentionnées au Chapitre 9 (notamment l'examen des flags DF, du TTL, de la taille de la fenêtre, des types d'options, des valeurs et de leur ordre, etc.) pour identifier le pare-feu.

Il existe également une autre possibilité, fondée sur la spécification RFC 1122[7] :

> 4.2.2.12 RST Segment : RFC-793 Section 3.4
>
> TCP DOIT permettre à un segment RST de contenir des données.
>
> DISCUSSION : il a été suggéré qu'un segment RST pourrait contenir du texte ASCII qui code et explique la cause du RST. Aucune norme n'a encore été établie pour ce type de données.

Et, en effet, même si aucune norme n'a été établie, certains systèmes choisissent de répondre aux paquets ACK errants par des messages RST verbeux (bien que souvent obscurs), en espérant que l'autre partie se trouve rassurée de savoir ce qui n'allait pas. Ces réponses contiennent souvent des mots clés internes ou, semble-t-il, des tentatives d'humour geek spécifiques au système d'exploitation, tels que `no tcp`, `reset`; `tcp_close, during connect` (Mac OS); `tcp_fin_wait_2_timeout`; `No TCP` (HP / UX); `new data when detached`; `tcp_lift_anchor, can't wait` (SunOS).

Si nous voyons un tel paquet RST verbeux envoyé à l'hôte en réponse à des problèmes de réseau ou à du trafic inattendu et si nous savons par ailleurs que le système distant d'où il semble provenir n'utilise pas ce genre de messages verbeux, nous pouvons en déduire qu'il existe un périphérique entre nous et le destinataire, probablement un pare-feu à états. Nous pouvons également dire quel est son système d'exploitation en comparant la réponse et les messages couramment produits par des systèmes d'exploitation communs et moins communs.

Ces deux techniques de fingerprinting se révèlent très efficaces pour détecter la présence de filtres de paquets à états chaque fois que le trafic réseau peut être observé lorsque surviennent des problèmes réseau à court terme. Ces techniques peuvent également être utilisées pour le fingerprinting actif sans devoir cibler le pare-feu lui-même mais en envoyant un paquet ACK errant vers une cible pour différencier les filtres sans états des filtres à états. En fonction de la manière dont la cible répond au paquet, l'attaquant peut alors définir la meilleure méthode d'approche du pare-feu (ou utiliser ces connaissances dans un autre but).

Fiabilité ou performances : la controverse sur le bit DF

Le Path MTU Discovery (PMTUD) est un rendez-vous pour le fingerprinting qui est étroitement lié au système destiné à éviter la fragmentation IP décrite au Chapitre 9.

Les versions récentes du noyau Linux (2.2, 2.4, 2.6) et de Windows (2000 et XP) implémentent et activent le PMTUD par défaut. Ainsi, à moins que ce paramètre ne soit modifié, tout le trafic provenant de ces système contient un bit DF (*don't fragment*). Là encore, l'algorithme de découverte du chemin a tendance à provoquer des problèmes dans certaines situations rares, mais qui se sont déjà produites.

Cas d'échec du Path MTU Discovery

Le problème de PMTUD vient qu'il repose sur la capacité qu'a l'expéditeur d'un paquet à recevoir le message d'erreur ICMP "Le paquet doit être fragmenté, mais paramétré DF" et à déterminer les réglages optimaux pour une connexion. Le paquet qui a déclenché

le message est supprimé avant d'atteindre sa destination et doit donc être redimensionné et envoyé à nouveau.

Si l'expéditeur ne reçoit pas ce message, il n'est pas au courant que ses paquets ne sont pas parvenus à destination. Cela entraîne au mieux un retard ou au pire le blocage de la connexion pour une durée indéfinie, car les retransmissions ne sont pas non plus susceptibles de traverser une liaison qui autorise une taille maximale pour les paquets inférieure à celle des paquets que l'expéditeur tente de transmettre.

En revanche, il n'est pas garanti que le message ICMP généré lorsque la taille d'un paquet est trop importante pour une liaison parvienne à l'expéditeur. Dans certains réseaux, à la suite d'une tentative mal conçue d'améliorer la sécurité, tous les messages ICMP sont simplement supprimés. Enfin, même si un périphérique envoie un message, il peut très bien ne jamais être livré.

Pourquoi supprimer les messages ICMP ? Historiquement, nombre de ces messages étaient connus pour causer des problèmes de sécurité : certains paquets ICMP de taille exceptionnelle ou fragmentés corrompaient la mémoire du noyau dans de nombreux systèmes (cette attaque est connue sous le nom de "ping de la mort"). Les messages ICMP envoyés aux serveurs de diffusion ont également été utilisés pour déclencher une tempête de réponses vers une fausse adresse source (attaques par réflexion ou attaque "Smurf"), ainsi que pour effectuer des attaques par déni de service (DoS). Des systèmes mal configurés interprétaient également souvent les messages du routeur*, un type spécifique de diffusion ICMP, comme une commande visant à modifier leurs paramètres réseau. Comme ils l'acceptaient sans se préoccuper si ces messages étaient dignes de confiance ou non, cela a ouvert une nouvelle voie intéressante pour les attaques. Si bien qu'aujourd'hui beaucoup craignent et bloquent ICMP.

NOTE

Des guides de sécurité suggèrent souvent de rejeter tout trafic ICMP et certains administrateurs système suivent ce conseil naïf. Je l'ai même vu dans une recommandation de test d'intrusion professionnelle écrite par un spécialiste renommé dont je ne peux malheureusement pas révéler le nom ici.

Un autre problème qui peut rendre le PMTUD peu fiable, c'est que certains messages d'erreur reçus proviennent de périphériques qui utilisent un espace d'adressage privé. Parfois, afin de limiter l'espace d'adressage IP public (qui est généralement coûteux),

* Les annonces des routeurs ont été conçues pour permettre la configuration automatique des hôtes du réseau sans avoir à entrer les paramètres manuellement. Le routeur diffuse périodiquement ou à la demande un message disant en gros : "Je suis là, utilisez-moi." Par défaut, quelques systèmes acceptaient des messages non sollicités sans trop d'hésitation, ce qui est évidemment une mauvaise idée.

les interfaces sur le câble qui relient le routeur et le pare-feu d'un réseau distant sont choisies parmi un pool d'adresses réservées à une utilisation locale et privée, au lieu de partir de celles effectivement routées vers ce réseau particulier depuis le monde extérieur.

Malheureusement, l'utilisation de l'espace d'adressage privé peut interrompre le PMTUD. Pourquoi ? Parce que, si un paquet en provenance du monde extérieur est trop volumineux pour être relayé à destination par le pare-feu du destinataire, le pare-feu envoie un message d'erreur ICMP avec l'adresse source du pare-feu lui-même, qui appartient au pool privé. Le pare-feu de l'expéditeur du paquet original peut alors rejeter le paquet de cette réponse car il semble venir du monde extérieur mais a une adresse IP d'un pool privé (peut-être même du même pool que le réseau LAN privé de l'expéditeur). Le pare-feu rejette ce trafic, car c'est souvent le signe d'une tentative visant à usurper l'identité d'un hôte interne de confiance. Toutefois, dans ce cas, cette décision interrompt le mécanisme de découverte PMTU relativement récent et l'expéditeur original ignore que ses paquets ne sont pas passés.

Pour aggraver les choses, même si toutes les conditions sont réunies et que le paquet atteint sa destination finale, de nombreux périphériques actuels limitent les taux de réponse ICMP et n'envoient qu'un certain nombre de messages pendant une période donnée. Cela a aussi été implémenté par mesure de sécurité. Comme les messages ICMP ont été conçus uniquement à des fins d'information et n'étaient pas essentiels à la communication avant l'introduction des algorithmes PMTUD, limiter le taux semblait être un bon moyen d'écarter certains types de dénis de suivi ou les attaques par épuisement de la bande passante.

La lutte contre PMTUD et ses retombées

À la lumière de ce qui précède, certains considèrent le PMTUD comme une conception assez mauvaise. Il offre une légère amélioration des performances, mais au prix de problèmes rares mais récurrents et généralement difficiles à diagnostiquer qui peuvent empêcher les utilisateurs d'accéder à certains serveurs ou bloquer leur connexion de manière inattendue. Bien que de nombreux algorithmes "de détection des trous noirs " aient été conçus pour détecter les hôtes ou les réseaux sur lesquels le PMTUD devrait être désactivé (ces travaux ont connu plus ou moins de succès), cela ne résout pas totalement le problème et peut introduire des retards supplémentaires – le plus souvent lorsque cela est le moins souhaitable.

Pour résoudre ces problèmes et éviter les plaintes, certains fournisseurs de pare-feu configurent leurs solutions pour qu'elles effacent le flag DF sur tout le trafic sortant. Cette légère modification est souvent appréciée mais elle offre aussi un excellent moyen d'identifier la présence d'un dispositif de filtrage et de réécriture des paquets. Si on observe sur une adresse ou un réseau donné les caractéristiques des systèmes où le

PMTUD est activé mais que les paquets entrants n'aient pas de flag DF comme prévu, on peut déduire la présence et le type d'un pare-feu et donc obtenir de petits bits de données sans interagir avec la victime.

Matière à réflexion

Ainsi s'achève ma petite histoire visant à montrer qu'améliorer les pare-feu et les rendre plus à même d'empêcher les intrusions et la reconnaissance directe les rendent également plus faciles à examiner par évaluation indirecte. Mais permettez-moi une brève parenthèse.

La découverte la plus bizarre et la plus intéressante est peut-être celle que j'ai faite en 1999. Bien que n'étant pas directement liée à la conception des pare-feu, elle n'en fournit pas moins une piste de réflexion intéressante pour tous ceux qui s'intéressent au problème du fingerprinting passif des systèmes intermédiaires.

Jacek P. Szymanski, avec qui j'ai travaillé brièvement et avec qui j'ai ensuite eu le plaisir de discuter de certains motifs inhabituels et suspects du trafic réseau*, a noté une augmentation soudaine du nombre de paquets TCP/IP cassés arrivant au port 21 536 (et, dans une moindre mesure, à des ports comme 18 477 ou 19 535). Ces paquets cassés provenaient toujours de ports comme 18 245, 21 331 ou 17 736 et avaient pour origine un grand nombre de systèmes dans l'espace d'adressage téléphonique exploité par Telekomunikacja Polska, l'opérateur téléphonique national en Pologne.

Une fois quelques-uns de ces paquets capturés, le trafic était étrangement et sévèrement altéré. Les paquets arrivaient avec des en-têtes IP en place (avec un type de protocole défini sur TCP), mais les en-têtes étaient immédiatement suivis de la charge TCP ; les en-têtes TCP avaient tout simplement disparu. La combinaison de ports observée provenait de l'interprétation des quatre premiers octets de la charge comme une paire de numéros (s'il y avait eu un en-tête TCP, ces numéros correspondraient à la combinaison des ports source et de destination). La paire 18 245 et 21 536 était simplement une représentation de la chaîne de texte "GET", les quatre caractères qui commencent la plupart des requêtes HTTP transférées sur le réseau. De même, 18 477 et 21 331 représentaient SSH-, la phrase d'ouverture de toute session Secure Shell. Et 19 535 et 17 736 représentaient EHLO, la commande qui ouvre toutes les sessions ESMTP (Extended SMTP).

---

* Une coopération qui, à un moment donné, a abouti à la création d'un groupe informel de chercheurs polonais qui, de 1999 à 2000, ont cherché à mettre en corrélation, à suivre et à expliquer de nombreux types bizarres et inattendus de modèles de circulation du trafic à travers le réseau.

Mais la raison pour laquelle ce type de trafic fit soudainement son apparition restait un mystère. En outre, pourquoi ne provenait-il que de ce réseau en particulier ? Et pourquoi ce type d'altération des paquets ne donnait-il pas lieu à des problèmes de connexion ou à d'autres désagréments pour les utilisateurs, si certains équipements du réseau les produisaient effectivement ?

La réponse suivit bientôt. Il s'est avéré que tout le trafic observé provenait des périphériques Nortel CVX, un système d'accès par modem que cet opérateur avait commencé à utiliser. Le problème n'apparaissait que sporadiquement, sous de fortes charges. Par conséquent, seul un faible pourcentage de paquets incomplets était envoyé et seul ce petit nombre parvenait aux destinataires (à leur plus grande surprise). Ce problème était sans doute dû à une gestion incorrecte de la mémoire tampon ou du verrouillage de la file d'attente, ce qui explique pourquoi il pouvait être remarqué que lorsque de nombreuses sessions étaient traitées presque simultanément. Certains paquets semblaient alors être envoyés trop tôt, alors qu'ils étaient encore "en construction" ou étaient altérés par l'implémentation.

La société corrigea son implémentation TCP/IP peu de temps après son déploiement en Pologne, et tout se termina bien. Mais, comme vous pouvez l'imaginer, ils n'étaient ni les premiers ni les derniers à laisser accidentellement une empreinte unique de leurs systèmes dans des paquets dont ils géraient le trafic.

Une fois de plus, la morale de cette histoire est qu'il est naïf de ne pas tenir compte de ce que nous ignorons généralement. Dans le monde en réseau actuel, les indices les plus ténus et les observations inhabituelles, inattendues ou inexpliquées sont extrêmement précieux. Ils sont faciles à trouver, mais difficiles à analyser.

Les différentes méthodes déployées pour déjouer le fingerprinting du système constituent peut-être aussi matière à réflexion et méritent d'être explorées. Plusieurs fournisseurs de pare-feu ont tenté d'incorporer des mesures antifingerprinting qui altèrent certaines caractéristiques des paquets en modifiant différents paramètres TCP/IP (comme les identifiants IP, les numéros de séquence TCP, etc.). Inutile de dire que cette solution aide en fait l'attaquant et produit un résultat exactement inverse de celui attendu : à moins que toutes les caractéristiques que le fingerprinting peut découvrir soient changées et rendues homogènes (y compris les numéros de séquence, les temps de retransmission, les valeurs de timestamp et ainsi de suite), il est non seulement possible de détecter le système d'exploitation sous-jacent mais aussi le pare-feu utilisé pour protéger le réseau.

C'est la vie !

12

Fuite des données de la pile

Encore un court récit expliquant où trouver
ce que nous n'avions pas du tout l'intention d'envoyer.

Parfois, un peu de chance suffit pour trouver des indices légers mais très utiles sur les personnes qui partagent le même réseau que vous et où elles se trouvent. Ce fut du moins le cas lorsque je découvris en 2003 un vecteur de divulgation d'informations assez intéressant et très évasif après plusieurs semaines de chasse intensive.

Le serveur de Kristjan

Mais reprenons dans l'ordre. Il y a plusieurs années, j'ai demandé à Kristjan, un ami, de me laisser utiliser un peu d'espace disque sur une de ses machines afin d'héberger certains de mes projets sur un système fiable et rapide. Il a accepté et, peu de temps après, j'ai commencé à transférer progressivement la plupart de mes programmes et documents vers leur nouvelle demeure. Parmi les projets que je transférais se trouvait une nouvelle version de p0f, mon outil de fingerprinting passif du système d'exploitation (dont j'ai parlé au Chapitre 9). Cet utilitaire implémentait des techniques d'analyse passive intéressantes mais, pour être vraiment puissant, il devait s'accompagner d'une large base de données de signatures des systèmes d'exploitation qui soit actualisée.

La mise à jour manuelle de la base était difficile à effectuer, et j'ai rapidement manqué de systèmes obscurs à ajouter.

Alors que recueillir des signatures pour les logiciels de fingerprinting actif nécessitait souvent d'interagir de façon répréhensible avec la cible (ce qui soulève des controverses, sature la liaison réseau et parfois écrase particulièrement les piles TCP/IP mal implémentées), le fingerprinting passif n'exigeait heureusement pas de mener ce genre d'action et pouvait être pratiqué sans effort sur tous les systèmes qui se connectaient au système de Kristjan pour afficher ma page. Pour encourager les visiteurs à participer, j'ai créé une page annexe sur laquelle ils pouvaient immédiatement voir leur empreinte, corriger les informations recueillies sur leur système ou ajouter une nouvelle signature. Cette page s'est révélée être un excellent moyen de recueillir des signatures et d'améliorer le logiciel, mais l'histoire ne se termine pas là.

Les événements prirent alors une tournure étrange. Kristjan décida d'héberger un autre site, à but lucratif celui-là, de manière à financer son système. Ce site, comme vous pouvez l'imaginer, ne traitait du tout de la sécurité des réseaux, du jardinage ou de quelque autre cause aussi noble. Au contraire, il était consacré à un aspect de notre vie moins prestigieux mais peut-être plus attrayant : le sexe, la nudité, et tout ce qui s'y rapporte. Cela me réjouissant, comme tout geek qui se respecte, non pas en raison du contenu de ce site mais parce qu'en quelques heures des millions de signatures de connexion à analyser par le logiciel que je développais commencèrent à affluer. Alléluia !

Des découvertes surprenantes

Mieux vaut prévenir que guérir : lors de l'élaboration du nouveau code de p0f, je décidai d'implémenter un certain nombre de contrôles afin de détecter dans le trafic entrant des motifs inconnus, bizarres ou improbables afin d'englober toutes les combinaisons dénuées de sens ou illégales des paramètres TCP/IP. Tandis que je pensais ne jamais rencontrer de paquets dont les paramètres soient bizarrement altérés (du moins pas en communiquant avec des systèmes courants et donc bien testés), il semblait n'y avoir aucun mal à implémenter cette fonctionnalité. De plus, s'il s'avérait effectivement qu'un système envoyait des paquets présentant un type d'anomalie en particulier, la capacité de le détecter constituerait une excellente façon de distinguer ce système d'exploitation des autres implémentations similaires qui ne partageaient pas ce défaut.

Pendant les mois heureux où cette tempête bénie de signatures eut lieu, j'ai vu les choses les plus étranges. J'ai finalement réussi à en expliquer certaines et à les documenter pour p0f, mais d'autres restaient un mystère. La plupart des contrôles d'anomalie que j'avais implémentés auparavant remplissaient leur rôle, si bien que je localisais immédiatement les systèmes qui partageaient les implémentations TCP/IP les plus inhabituelles. Mais

une chose était particulièrement préoccupante et difficile à croire, si bien que je décidai d'y accorder plus d'attention.

Deux des tests – une vérification de la valeur ACK définie dans les en-têtes TCP/IP lorsque le flag ACK n'est pas défini (une action futile, en effet) et un test de la valeur URG définie lorsque le flag n'est pas activé – semblaient relativement vides de sens au premier abord et ne donnaient jamais de résultats intéressants, jusqu'à ce que je remarque quelque chose de tout à fait inhabituel. Je notai que certains systèmes sous Windows 2000 et XP qui se connectaient au serveur de Kristjan avaient de temps à autre des valeurs URG ou ACK différentes de zéro dans des paquets où aucun de ces flags n'était défini (en particulier dans les paquets SYN qui ouvrent une nouvelle connexion).

Avoir des valeurs URG ou ACK définies alors que leurs flags respectifs ne le sont pas ne représente pas un problème à strictement parler. Selon la spécification RFC793, ces valeurs perdent simplement toute signification lorsque c'est le cas. Par exemple :

> Pointeur d'urgence : 16 bits
>
> Ce champ communique la valeur courante du pointeur d'urgence comme un offset positif du numéro de séquence dans ce segment. Le pointeur d'urgence pointe vers le numéro de séquence de l'octet qui suit les données urgentes. Ce champ doit seulement être interprété dans les segments où le contrôle URG est défini.

Dans sa façon toute particulière de présenter les choses, la spécification RFC793 nous dit que cette anomalie ne risque pas de provoquer de problèmes réseau. Elle aurait donc pu passer inaperçue indéfiniment. Et je ne la remarquai tout simplement que parce qu'il s'agissait d'une bizarrerie.

J'ai d'abord pensé que la faute venait d'un élément particulier de l'équipement réseau, comme pour la plupart des problèmes décrits au Chapitre 11, mais ce n'était pas le cas. Les anomalies provenaient de systèmes simples, non de réseaux entiers, et n'étaient pas persistantes ; elles apparaissaient seulement dans quelques paquets (avec des valeurs fixes ou changeant de façon aléatoire), puis disparaissaient sans jamais réapparaître de nouveau sur les connexions suivantes. Ce problème semblait être l'apanage unique de Windows ; aucun système d'exploitation minoritaire n'apparaissait dans l'ensemble des systèmes présentant ce défaut.

J'ai passé plusieurs semaines à tenter de trouver l'origine du problème. J'ai déployé d'autres installations dans des environnements plus contrôlés. À ma grande surprise, le problème est apparu, même dans des réseaux locaux et même dans les systèmes les plus à jour, mais seulement pour de courtes périodes de temps. Les utilisateurs ne se souvenaient pas avoir fait quoi que ce soit d'inhabituel lorsque ce type de trafic provenait de leur système et je ne trouvais pas de type particulier de communication ni d'action qui aurait pu déclencher cela ; il semblait n'y avoir aucun motif.

Troublant.

Révélation : la reproduction du phénomène

J'étais sur le point d'abandonner. Je publiai mes observations sur plusieurs listes de diffusion publiques (notamment VULN-DEV, une liste de discussion populaire sur la vulnérabilité hébergée par Security Focus), à la recherche d'une analyse plus approfondie et de l'avis d'autres chercheurs, mais cela ne donna aucun résultat. Et puis je surpris par chance une de mes propres stations de test en train de reproduire exactement ce schéma de comportement alors que je travaillais sur un problème complètement différent. Il se trouve que j'avais un sniffer s'exécutant en tâche de fond (n'est-ce pas le cas pour nous tous ?).

Je pus bientôt effectuer un diagnostic : le problème survenait lorsque la station de travail réalisait un transfert de fichier en tâche de fond ou une autre opération sollicitant beaucoup le réseau tout en essayant d'établir une connexion. Dans presque tous les systèmes d'exploitation, le paquet à envoyer est d'abord créé dans la mémoire principale du système, à l'aide soit d'une mémoire tampon statique (un lieu fixe dans la mémoire utilisé exclusivement dans ce but) soit d'une mémoire tampon dynamique (une partie de la mémoire allouée selon les besoins et pouvant utiliser la mémoire déjà utilisée auparavant pour d'autres fins). Dans ce scénario particulier, lorsque deux connexions se produisaient à peu près simultanément, la mémoire tampon utilisée pour créer les paquets sortants avant de les envoyer à la carte réseau semblait ne pas être correctement initialisée avant d'être utilisée. En fait, tout le contenu restant des utilisations précédentes de la mémoire tampon n'était pas effacé ; autrement dit, la mémoire tampon n'était pas effacée parce qu'elle avait été utilisée dans un but différent la fois précédente. Le code d'implémentation suppose que tout le contenu de la mémoire tampon est à zéro et ne prend pas la peine de toucher le contenu qu'il n'a pas besoin d'initialiser à une valeur particulière (comme c'est le cas pour les valeurs URG et ACK lorsque leurs flags respectifs ne sont pas définis). Par conséquent, une partie du contenu résiduel est envoyé dans le paquet.

Naturellement, tous les autres champs IP et TCP étaient correctement initialisés ; seuls les champs URG et ACK ne l'étaient pas puisqu'ils n'avaient pas d'importance dans ce contexte particulier. Mais cette omission signifiait qu'une petite partie des données appartenant à une connexion différente (ou à un aspect différent des opérations de l'ordinateur) était envoyée à une autre partie. Le problème se manifestait uniquement lorsque plusieurs sessions étaient ouvertes (ce qui est courant lorsqu'on navigue sur le Web ou qu'on télécharge en tâche de fond, par exemple), mais pas lorsque le système était inactif.

L'importance de l'information divulguée dans cette situation est double :

- On peut considérer qu'il s'agit d'un scénario de divulgation d'informations classique. Bien que la quantité d'informations divulguées dans chaque paquet où les valeurs URG et ACK ne sont pas initialisées correctement soit assez faible et puisse

ne pas être significative (sauf si la mémoire tampon contenait quelque chose d'intéressant), elle peut être utile dans certains cas de figure, surtout lorsqu'une session simultanée pouvant contenir des informations sensibles et donc le bogue lui-même peut être provoquée par une entité externe.

- Cette vulnérabilité peut être considérée comme une mesure de fingerprinting pratique qui révèle des informations supplémentaires sur le système d'exploitation et son état – une façon simple de différencier les systèmes qui utilisent largement le réseau de ceux qui sont inactifs.

Voilà. Même si la signification de cette découverte ne doit pas être surestimée, j'ai décidé de l'inclure ici pour sa valeur divertissante et pour montrer à quel point il est facile d'obtenir des données complexes d'une partie distante sans même le demander.

Matière à réflexion

Il est facile de rejeter la faute sur les développeurs. Bien qu'ils soient bien sûr responsables de ne pas initialiser la mémoire correctement, le fait même d'avoir un "activateur" séparé pour un champ de l'en-tête tient peut-être plus à un défaut de conception du protocole TCP lui-même et peut contribuer à ce genre de problème. Nous avons vu au Chapitre 7 certains petits défauts similaires des spécifications de protocole, lorsque nous avons parlé d'un type semblable de vulnérabilité dû au simple fait de suivre une spécification trop à la lettre sans beaucoup réfléchir aux effets secondaires potentiels.

13

Fumée et miroirs

Ou comment disparaître avec élégance.

Beaucoup de cas de divulgation d'informations examinés jusqu'ici nécessitent qu'une analyse minutieuse des données transmises par un système distant soit effectuée pour déduire certains faits sur l'expéditeur ou pour intercepter plus de données qu'il n'a conscience d'en envoyer. Dans plusieurs cas, toutefois, on ne peut obtenir qu'une preuve circonstancielle de la présence d'une certaine forme d'activité.

Comme indiqué aux Chapitres 1 et 2, en interprétant précisément cette preuve on peut déterminer l'endroit probable où se trouve l'utilisateur ou une application qui traite les données sensibles et ainsi révéler indirectement les secrets de la machine de la victime sans avoir à accéder aux données elles-mêmes.

En raison de certaines caractéristiques du protocole IP, bon nombre de ses implémentations sont vulnérables et divulguent des éléments de cette preuve circonstancielle, d'une manière assez semblable à celle que nous avons vue précédemment avec certains types de générateurs de nombres pseudo-aléatoires ou certains algorithmes de traitement des données plus ou moins complexes. Il peut être intéressant d'observer puis de déchiffrer soigneusement ces informations car elles n'offrent dans le pire des cas plus de renseignements que nécessaire sur les habitudes d'un concurrent en général ou sur ses activités en particulier.

Jusqu'à présent, nous nous sommes consacrés dans ce livre aux attaques sur la couche IP, ce qui nécessite d'observer directement le trafic en provenance d'un expéditeur, même sans interagir en général avec la victime. Dans ce chapitre, en revanche, nous allons examiner une attaque IP active spectaculaire bien qu'indirecte dans laquelle un attaquant établit le profil de ses victimes en devinant ce qu'il ne peut pas voir. Pour cela, il interagit avec une innocente machine victime qui n'est pas le vrai sujet du test sans qu'elle ne le sache et sans son consentement pour apprendre ce qu'il est possible de l'être sur la victime réelle de l'attaque.

Une telle approche ne semble pas le moyen le plus simple de recueillir des données. Donc, dans l'esprit d'un geek, pourquoi ne pas prendre la route touristique, même si elle est plus longue, et examiner le paysage en détail ?

L'usurpation d'adresse IP : le scan de port avancé

Les internautes malveillants utilisent fréquemment le balayage des ports pour préparer leurs attaques et réaliser le fingerprinting du système. Lors du balayage des ports, un attaquant potentiel tente de se connecter pendant une courte durée à chaque port sur un système et à établir la carte de tous les programmes qui écoutent le trafic réseau. De cette manière, il peut savoir où attaquer en trouvant tous les services réseau sur le système qui sont vulnérables ou potentiellement intéressants. Dans de nombreux cas, il peut aussi déterminer quel système d'exploitation utilise sa victime, car les services par défaut sont souvent spécifiques à chaque système d'exploitation.

Le premier problème du balayage de port traditionnel est qu'il est assez bruyant – la victime est susceptible de remarquer une tempête ou même un flux régulier de tentatives de connexion à des ports inhabituels. Il n'est pas non plus facile de se cacher ; l'attaquant doit être en mesure de voir les réponses à ses paquets SYN pour savoir si un port est ouvert ou fermé. Les ports ouverts répondent avec SYN+ACK, les ports fermés, avec RST, tandis que les ports filtrés par un pare-feu sont susceptibles de ne générer aucune réponse ou un message ICMP (*Internet Control Message Protocol*). En conséquence, l'attaquant ne peut pas simplement usurper l'adresse source sur tous les paquets sortants ; il doit révéler son identité en fournissant des adresses sources qui sont routées en retour vers le réseau qu'il écoute pour détecter le trafic entrant.

L'arbre qui cache la forêt

Qu'on effectue le balayage de port par curiosité (pour voir par exemple quel système d'exploitation un concurrent utilise) ou pour préparer une tentative d'attaque, on souhaite en général laisser aussi peu de traces que possible et éviter d'alerter la victime. Les administrateurs réseau et les autorités ont généralement une idée assez négative du

balayage des ports des hôtes et des réseaux. Bien que la question de savoir si ces analyses doivent être considérées comme malveillantes ne soit toujours pas tranchée, la personne qui effectue cette découverte perd presque toujours quand un administrateur système agacé décide de signaler un abus ou quand un employé qui essaie de sonder les réseaux de la concurrence se fait remarquer, quels que soient les véritables intentions et les plans de ce testeur curieux.

Pour camoufler le balayage de port, on déploie couramment un balayage "leurre". L'attaquant envoie alors des paquets SYN à chaque port depuis un certain nombre de fausses adresses, ainsi que depuis sa véritable adresse IP. La victime traite ces faux paquets de la même façon que les vrais, mais les réponses à ceux qui sont faux sont bien sûr envoyées dans le vide. En conséquence, il est beaucoup plus difficile pour la victime de savoir qui est réellement derrière le scan, car elle doit effectuer une analyse approfondie ou procéder par tâtonnements pour éliminer tous les systèmes leurres de la liste des paquets source. Pourtant, avec un peu d'effort, il est possible de localiser l'expéditeur sans faire appel aux autorités, même si l'attaquant espère décourager la victime en rendant un tel incident mineur trop long à résoudre entièrement.

L'idle scan

Comme c'est souvent le cas, une personne qui avait trop de temps libre et qui le perdait à lire des spécifications de protocole au lieu de faire quelque chose de productif découvrit la défense ultime contre le risque d'être découverte. Ainsi naquit la technique baptisée "idle" scan (balayage "inactif"). Initialement conçue par Salvatore "antirez" Sanfilippo en 1998, elle a rapidement été implémentée à grande échelle et est devenue très populaire parmi les hackers (à la fois les simples curieux et les personnes malveillantes)[1].

L'idle scan se fonde sur une observation importante. Pour citer la spécification RFC793 :

> En règle générale, un paquet reset (RST) doit être envoyé chaque fois qu'arrive un segment qui n'est apparemment pas prévu pour la connexion en cours. Un paquet RST ne doit pas être envoyé s'il n'est pas certain que ce soit le cas.

Les paquets RST du protocole TCP sont utilisés pour clore une connexion sans condition et dire à l'expéditeur de cesser toute nouvelle tentative de communiquer. Le système, sans beaucoup d'hésitation, envoie un paquet RST lorsqu'il rencontre du trafic imprévu, selon la règle de la RFC793 (naturellement, les paquets RST eux-mêmes, même lorsqu'ils ne sont pas attendus, ne reçoivent aucune réponse ; s'ils l'étaient, un flux interminable de paquets RST rebondirait au moindre hoquet du réseau).

L'idle scan use et abuse habilement du fait que la machine témoin gère tous les paquets inattendus de cette façon. L'attaque permet aux membres malveillants du réseau de scanner une victime avec laquelle ils n'ont pas l'intention de communiquer directement.

Lors de l'idle scan, l'attaquant choisit au hasard un système sur Internet et l'utilise sans qu'il ne se doute de rien pour scanner un troisième système (la vraie victime), sans jamais dévoiler son identité.

L'idle scan fonctionne de la façon suivante : l'attaquant usurpe l'identité d'un paquet SYN destiné à un port qu'il veut contrôler sur le système de la victime. Ce paquet est adressé à la victime, mais l'adresse de retour indique l'adresse du système témoin au lieu de celle du système de l'attaquant. Cette technique à elle seule ne semble pas mener bien loin, mais attendez la suite.

Ce qui se passe ensuite dépend si le port est ouvert ou non :

- Si le port sondé sur le système de la victime répond au système témoin par un paquet RST, la machine témoin reçoit et conserve simplement le RST en silence, sans générer aucun trafic en retour vers la victime.
- Si le port sondé est ouvert, la victime répond par un message SYN+ACK. Le témoin, incrédule, conclut qu'il n'a jamais envoyé de paquet SYN auparavant et envoie donc un message RST pour indiquer à la victime qu'elle se trompe manifestement et qu'elle ferait mieux de s'arrêter maintenant. La victime, en bon mouton de panurge, accepte la réponse et abandonne tous les enregistrements de la connexion qu'elle espérait accepter.

L'importance de cette distinction est difficile à apprécier en premier lieu. Mais souvenez-vous du Chapitre 9 et des informations sur un des champs de l'en-tête IP :

> Le numéro d'identification (ID) est une valeur 16 bits qui différencie les paquets IP lorsque la fragmentation se produit. Sans IP ID, si deux paquets sont fragmentés simultanément, les fragments de ces deux paquets seront altérés, mélangés ou endommagés au réassemblage. Les IP ID identifient de façon unique plusieurs mémoires tampon de réassemblage pour les différents paquets. La valeur utilisée pour cela est souvent choisie simplement en incrémentant un compteur à chaque paquet envoyé ; le premier paquet envoyé par un système a un IP ID de 0, le deuxième, un IP ID de 1, et ainsi de suite.

Comme l'attaquant a choisi une machine victime qui utilise en effet ce système de sélection d'IP ID (et il a l'embarras du choix), il peut désormais facilement savoir si la machine victime a envoyé un paquet IP dans un délai donné. Pour cela, il envoie tout simplement du trafic dénué de sens au système témoin avant et après le sondage et compare les valeurs des IP ID dans les réponses qu'il reçoit. S'il observe que deux IP ID ne diffèrent que de 1, cela signifie qu'aucun paquet n'a été envoyé par la machine témoin entre-temps. En revanche, si l'écart est supérieur à 1, cela indique que certains paquets ont effectivement été échangés, même si l'attaquant ne peut pas savoir avec certitude avec qui a eu lieu cet échange.

L'attaquant peut également effectuer un sondage juste avant et juste après l'envoi d'un paquet usurpé à la victime. Les réponses de la machine témoin lui permettent ainsi de savoir si un port est ouvert ou fermé. Si l'IP ID du témoin augmente, il est probable qu'il réponde à la victime avec un paquet RST, ce qui signifie que la victime doit avoir auparavant envoyé un message SYN+ACK en réponse au paquet usurpé. L'attaquant peut alors conclure que ce port est ouvert. Si, en revanche, le témoin produit le prochain IP ID comme prévu, il n'a rien reçu de la victime ou a décidé d'ignorer les paquets RST reçus.

Il y a, bien sûr, certaines considérations pratiques. Il est primordial que la machine témoin ait une activité relativement limitée pendant l'idle scan et que le test soit répété plusieurs fois afin d'éliminer les résultats faussement positifs, car sinon l'attaquant risque de mal interpréter certaines communications tierces du côté du témoin et penser qu'un port spécifique est ouvert sur la machine de la victime.

NOTE

Cependant, aucune de ces deux questions ne s'est révélée représenter un vrai problème et de nombreux utilitaires avancés implémentent l'idle scan et le font bien (depuis le précurseur idlescan en 1999 ou l'ingénieux NMAP actuellement).

L'importance de l'idle scan vient de sa capacité à masquer l'origine d'un balayage des ports non pas en essayant simplement de décourager la victime mais en rendant impossibles à identifier toutes les communications émanant de l'attaquant. Cela rend plus difficile de procéder au suivi de l'attaquant sans l'aide du propriétaire de la machine témoin (comme ce dernier peut se voir demander des IP ID par l'attaquant dans un cadre légitime, comme une session HTTP, il peut par conséquent être difficile de savoir s'il a été utilisé pour une attaque ou pas du tout) ou celle d'entités externes (autorités judiciaires et fournisseurs d'accès Internet). Comme, en général, les autorités déclenchent des poursuites judiciaires non parce qu'un système a été sondé mais uniquement lorsqu'il est compromis (les concurrents curieux peuvent dormir tranquilles) et qui nécessitent que la victime reconnaisse que son système a été compromis (ce qui n'est pas toujours facile pour certaines grandes entreprises), l'attaquant se sent plutôt en sécurité.

NOTE

Bien que les résultats obtenus ne semblent *a priori* pas différents de ceux obtenus avec un scan SYN classique, l'idle scan offre des perspectives de balayage des ports assez uniques. L'utilisation d'une machine témoin permet de voir le système de destination à partir de son point de vue. Si ce témoin dispose de privilèges d'accès supérieurs au système de la victime (si, par exemple, il s'agit d'un système situé à l'intérieur d'un réseau protégé par un pare-feu ou un système pour lequel certaines règles de filtrage IP sont définies de façon laxiste afin de faciliter l'accès à un réseau d'entreprise), on peut utiliser l'idle scan pour découvrir les rouages internes d'un réseau protégé.

Se défendre contre l'idle scan

Il n'existe à l'heure actuelle aucune défense immédiate contre l'idle scan ni aucun moyen simple de le différencier d'un scan SYN ordinaire. Toutefois, il est assez facile d'empêcher son système de devenir une machine témoin en utilisant des IP ID aléatoires ou constants, comme nous l'avons vu au Chapitre 9. Même si cela ne rendra pas plus difficiles les attaques menées contre vous ni les attaques en général (beaucoup de systèmes utiliseront toujours des identifiants séquentiels), cela empêchera votre réseau d'être abusé dans ce but précis.

Pour éviter une attaque qui contourne le pare-feu, faites preuve de bon sens lors de la conception des voies d'accès pour les systèmes extérieurs et utilisez le paramètre Ingress filtering approprié sur les systèmes passerelles pour interdire tous les paquets en provenance d'Internet dont les adresses sources semblent appartenir à un réseau protégé. Même si, comme nous l'avons vu précédemment, ce type de filtrage peut empêcher les mécanismes de PMTU (*path maximum transmission unit*), il résout généralement plus de problèmes qu'il n'en pose.

Matière à réflexion

Bien que ce soit moins facilement réalisable, il est toujours possible d'utiliser des IP ID pour établir le profil général de l'activité IP. En fait, lorsque la victime établit une session interactive avec un système distant, les IP ID peuvent même être utilisés pour analyser la frappe des touches dans le temps ou pour des actions analogues, ce qui rend donc cette technique applicable dans les scénarios d'attaque dans le temps déjà abordés. De même, vous pouvez améliorer les capacités de traçage de l'utilisateur en mesurant le nombre de paquets envoyés par un hôte en particulier entre deux visites consécutives à un réseau contrôlé.

En fonction de la conception du générateur ISN, vous pouvez également utiliser les numéros de séquence TCP sur certains systèmes pour disposer des mêmes fonctionnalités qu'offre l'analyse des identifiants IP. Je vous encourage à explorer cette idée plus en détail.

Pour ce qui est de traquer la source d'un idle scan (ou de toute autre attaque par usurpation), reportez-vous au Chapitre 17.

14

L'identification du client : vos papiers, s'il vous plaît !

Voir par le biais d'un déguisement mince peut se révéler utile dans de nombreux cas.

En local, il est assez simple de connaître la véritable identité d'un logiciel et sa légitimité sur l'ordinateur qui l'exécute. Mais il n'est pas si facile de le faire sur un réseau.

Les administrateurs système et les développeurs d'applications cherchent souvent à identifier le logiciel utilisé à l'autre extrémité lors d'une session réseau, et ils y parviennent avec plus ou moins de succès.

Ils essaient d'identifier les logiciels pour plusieurs raisons. Dans le cas du WWW (World Wide Web), il s'agit généralement d'optimiser le contenu destiné au client en fonction du moteur de rendu utilisé, que ce contenu soit légitime ou malveillant. Dans de nombreux autres systèmes de communication – messageries instantanées, clients mail, et ainsi de suite –, identifier les clients permet de garantir le respect de la politique réseau et de détecter des communications émanant d'applications éventuellement dangereuses ou inacceptables. Enfin, les programmeurs eux-mêmes tentent d'identifier les programmes pour éviter qu'un logiciel non approuvé (ou sans licence) n'utilise un service réseau en particulier (ce qui les prive d'une partie de leurs revenus) ou pour détecter de tels incidents et prendre des mesures correctives.

La manière la plus simple et la plus courante d'identifier l'autre partie repose sur l'examen des informations volontairement affichées par le système distant. Ces informations peuvent provenir d'une bannière de "bienvenue" fournie par un serveur, être obtenues en consultant les en-têtes de protocole envoyés par un client (le préfixe X-Mailer pour les e-mails, User-Agent dans les sessions WWW, etc.), en analysant l'état textuel et les messages d'erreur ou d'avertissement utilisés par le service pour répondre à certains types de trafics*. Malheureusement, la première méthode est extrêmement peu fiable et facilement sabotée par les utilisateurs qui ont quelque chose à cacher. La dernière méthode est intrusive et assez difficile à utiliser sur les clients sans causer de problèmes (la plupart des logiciels clients sont conçus pour abandonner et se plaindre à la première condition d'erreur qu'ils rencontrent ; les utilisateurs qui, à la suite d'une tentative d'identifier leurs logiciels, rencontrent un message d'erreur et ne peuvent pas accéder légitimement à un service n'accueilleront pas ce fait avec joie).

Camouflage

L'examen des annonces textuelles produites par le client n'est pas fiable, non seulement parce que les utilisateurs peuvent camoufler leur logiciel Internet (navigateurs Web, clients de messagerie, etc.) afin d'imiter les réponses des clients les plus fréquemment utilisés, mais aussi parce qu'ils ont souvent une bonne raison d'essayer, soit pour se fondre dans la masse soit tout simplement pour tromper les serveurs qui ont tendance à mieux connaître qu'eux la version d'un programme à utiliser. Ce camouflage est simple à réaliser, à l'aide d'une fonctionnalité intégrée du client ou en modifiant les sources ou les valeurs binaires d'un programme avec l'un des multiples utilitaires disponibles gratuitement.

En outre, comme de nombreux environnements d'entreprise ont commencé à implémenter un filtrage plus rigoureux du contenu afin de bloquer le trafic indésirable, certains codeurs qui travaillent sur des applications plus contestables se sont alors mis à usurper l'identité d'un logiciel inoffensif. Il n'y a pas si longtemps, les applications P2P de partage de musique, les chevaux de Troie malveillants et les logiciels espions ont commencé à se faire passer pour le navigateur Web le plus répandu, Microsoft Internet Explorer, dans leurs communications sortantes. Il en va de même pour beaucoup de robots utilisés par des entreprises commerciales assez douteuses dans le monde entier pour collecter les adresses Web.

---

* AMAP, de THC, est un outil populaire qui utilise le fingerprinting pour analyser les réponses. Pour en savoir plus, visitez l'adresse suivante : http://www.thc.org/releases.php. NMAP, créée par Fyodor et qui peut identifier des services en analysant des bannières.

D'autres protocoles sont également la proie des imitateurs. Il n'est pas surprenant que la majorité des logiciels de mailing de masse tant méprisés qu'utilisent les spammeurs et les escrocs feignent d'être des programmes comme Microsoft Outlook, PINE, Mutt, Eudora, The Bat! ou Netscape Mail. La règle de base est de se cacher derrière un camouflage pour échapper au contrôle des administrateurs réseau, qui, s'ils venaient à prendre conscience de la présence du logiciel, trouveraient facilement comment le bloquer. Aucun spammeur sain d'esprit n'annoncera que ses e-mails proviennent d'"Oncle Bernie, le sinistrement célèbre expéditeur de mailing de masse. Edition Ultime", tout simplement parce qu'il serait alors trop facile pour un utilisateur ou un filtre de spam de les détecter et de les supprimer.

Définition du problème

Puisqu'il est facile de modifier le texte de base des réponses et les bannières renvoyées par un programme, nous devons trouver un meilleur moyen de détecter les subterfuges qu'en utilisant des réponses textuelles simples correspondantes pour identifier le logiciel client avec une précision raisonnable. Des solutions qui se contentent de vérifier des paramètres ou des réponses moins évidents sont vouées à l'échec à un moment ou à un autre : bien qu'il soit possible dans presque tous les cas de concevoir un seul contrôle pour identifier un type particulier de logiciel indésirable, trois têtes vont repousser à la place de celle qui vient d'être coupée.

En pratique, il devient rapidement impossible de tenter de répondre à chaque incarnation de logiciels malveillants. Dans certains cas, une détection générale des clients malveillants peut être réalisée en vérifiant simplement les motifs qui sont clairement révélateurs de la nature des abus qu'on espère empêcher : contrairement au logiciel d'un spammeur, il est assez improbable qu'un client de messagerie légitime essaie d'envoyer 10 millions de courriers en même temps. Pourtant, cette approche est très limitée : alors qu'elle peut parfaitement fonctionner pour certains protocoles et certaines attaques clairement définis, il en va autrement pour le trafic WWW. Et il est difficile d'atteindre la bonne cible sans se retrouver avec un nombre excessif de faux résultats positifs ou sans oublier certains programmes.

Comme le WWW est considéré comme la base de tous les services Internet à la disposition des utilisateurs finaux, il est un des rares protocoles qui doit simplement être ouvert à presque tous. Par conséquent, le trafic Web est le plus souvent choisi par les applications malveillantes pour masquer leur comportement dans un système et camoufler les données qu'ils transfèrent à un hôte distant. Il n'est pas rare que les navigateurs Web déclenchent des tempêtes de connexions sur différents sites ou effectuent des milliers de requêtes par heure. En même temps, il est impossible d'envoyer des informations sensibles à un hôte distant en une seule connexion brève. Le profiling du trafic est donc loin de fournir une réponse dans ce cas.

Vers une solution

Compte tenu de tout cela, il semble extrêmement difficile de distinguer un logiciel espion ou un cheval de Troie d'une demande légitime. Toutefois, il s'avère que quelques bons utilitaires permettent d'identifier précisément ces types de programmes, ce qui permet aux parties intéressées d'identifier de façon plus précise les applications client. La solution la plus prometteuse et universelle, généralement considérée comme une *analyse comportementale* (un terme fantaisiste pour désigner l'étude des "motifs dans le temps"), vise à analyser les légères dépendances internes entre les portions consécutives du trafic, par opposition à l'observation des données effectivement échangées dans un seule requête ou du volume des connexions dans le temps. Comme ces dépendances sont étroitement associées aux algorithmes internes et aux performances d'un programme, elles sont beaucoup plus difficiles à usurper que la plupart des autres mesures qu'on peut examiner. Je vais discuter de cette approche dans ce chapitre et proposer une analyse simple des outils permettant d'atteindre ce niveau de précision et de détail, en utilisant le trafic du World Wide Web comme exemple.

Mais avant de nous plonger dans les détails, nous avons besoin d'un peu d'histoire. Examinons rapidement l'histoire du World Wide Web, la conception des clients Web et les protocoles qu'ils utilisent pour communiquer avec les serveurs. Tout a commencé plus tôt que vous ne le pensez…

Une (très) brève histoire du Web

Le concept de World Wide Web n'est pas particulièrement difficile à saisir : l'idée est de donner aux utilisateurs un accès instantané à un certain nombre de documents liés contenant des références croisées et qui intègrent différents types d'informations. Assez simple.

Le Web tel que nous le connaissons aujourd'hui est constitué principalement de texte accompagné de métadonnées (des références à d'autres fichiers, des éléments de mise en forme, des annotations et des éléments dynamiques ou interactifs), souvent mis en valeur avec du contenu multimédia (vidéos, musique, et applications diverses). Il représente l'esprit de notre époque et constitue une toute nouvelle façon de communiquer et de trouver des informations. Mais l'idée du Web n'est pas nouvelle. Elle existait depuis de nombreuses années avant que la technologie ne rende possible cet ensemble de fonctionnalités pour les documents électroniques, peut-être même longtemps avant que les documents électroniques ne soient considérés comme une possibilité sérieuse.

Selon un document[1] publié par le World Wide Web Consortium (W3C), le concept de lien hypertexte a été envisagé pour la première fois dans un article de la revue *Atlantic Monthly*[2] en 1945 par Vannevar Bush, qui était directeur de l'OSRD (Office of Scientific Research and Development) pendant et après la Seconde Guerre mondiale.

Bush a décrit un dispositif appelé Memex, une unité électromécanique personnelle qui peut en fait être considérée comme un prédécesseur des assistants personnels actuels. Memex offrait un stockage des fichiers personnels et des documents de l'utilisateur et visait à fournir des mécanismes intuitifs pour accéder à ces données. Memex était entre autres capable de créer et de suivre des liens entre les documents stockés sur microfilm. Mais l'idée d'un dispositif mécanique follement complexe fonctionnant sur microfilm n'a pas vraiment retenu l'attention à l'époque.

La notion de lien hypertexte a ressurgi à plusieurs reprises dans les années suivantes et les premiers essais d'implémentation dans les ordinateurs eurent lieu dans les années 1960. Cependant, ces tentatives ne furent pas particulièrement couronnées de succès, en grande partie parce que la puissance de calcul nécessaire pour que la technologie séduise les utilisateurs n'était pas encore disponible.

Ce moment arriva à la fin des années 1980. Après l'explosion des micro-ordinateurs et peu avant l'attaque frontale de la plate-forme PC, des membres du Conseil européen pour la recherche nucléaire (CERN)* envisagèrent les possibilités de créer des hyperliens. Tim Berners-Lee, l'un des chercheurs du CERN, est de toute évidence la personne officiellement coupable de la diffusion du HTML (*Hypertext Markup Language*), une série de contrôles pour l'intégration de métadonnées, de liens et de ressources multimédias dans les fichiers texte (à vrai dire, le HTML, qui est au cœur du Web tel que nous le connaissons, n'est guère une conception entièrement nouvelle et emprunte quelques idées au SGML (*Standard Generalized Markup Language*), décrit par la norme ISO 8879 en 1986). Le premier navigateur Web est né peu de temps après sur NeXT, une plate-forme informatique aujourd'hui peu connue mais qui était très innovante et avancée pour l'époque. Ce navigateur fut alors baptisé World Wide Web.

À partir du moment où ce nom accrocheur était trouvé, la révolution devint impossible à arrêter. En 1992, Tim Berners-Lee déposa une première spécification[3] du protocole HTTP (*Hypertext Transfer Protocol*), un outil qui encapsule les données HTML et d'autres ressources dans les communications serveur/client. En 1993, plusieurs moteurs de navigateur Web étaient disponibles et une poignée de serveurs Web offraient déjà leur contenu aux visiteurs curieux. Bien sûr, le HTTP ne représentait que 0,01 % de l'ensemble du trafic, mais il ne cessait d'augmenter !

---

* Laboratoire européen de physique des particules de Genève, Suisse.

Le premier navigateur Web populaire, Mosaic, fut développé au centre de recherche NCSA (National Center for Supercomputer Applications) de l'université de l'Illinois. Il empruntait le code de Berners-Lee mais ajoutait la prise en charge d'autres contenus que le texte et introduisait de nombreuses autres fonctionnalités qui nous semblent banales aujourd'hui. Le code de Mosaic a fini par évoluer pour devenir Mozilla, qui à son tour a servi de code de base pour Netscape Navigator (qui sera ensuite adapté dans le projet open source Mozilla, dont le code sera ensuite utilisé pour les générations suivantes de Netscape Navigator – simple, non ?). Au même moment, afin d'embrouiller davantage les utilisateurs, la société Spyglass transforma Mosaic pour créer la base de ce qui allait devenir le principal concurrent de Netscape : Microsoft Internet Explorer.

En 1994, le W3C, un organisme conçu pour superviser le développement du Web, fut fondé. La première version officielle de ce protocole, étendue et nettement améliorée, a été déposée par Tim Berners-Lee, Roy T. Fielding et Henrik Frystyk en 1996, bientôt suivie par les spécifications de code HTML 3.2. HTTP et HTML, désormais régis par le W3C et qui ont connu plusieurs améliorations au cours des années suivantes. Et vous connaissez tous la fin de l'histoire. À moins que ce ne soit que le début ?

Notions élémentaires sur le protocole de transfert hypertexte

HTTP[4] est un protocole fondé sur le texte étonnamment simple, construit par-dessus TCP/IP. Un client de ce protocole se connecte à un service où le protocole HTTP est activé sur un serveur distant et formule une requête, en demandant une ressource spécifique sur le serveur. La première ligne d'une requête HTTP contient les paramètres suivants :

- Une méthode d'accès à la ressource. Le plus souvent, le client demande simplement à récupérer un fichier, en émettant une requête GET (bien que d'autres méthodes existent pour envoyer les données d'un formulaire, effectuer des diagnostics, stocker des données sur un serveur ou exécuter certaines extensions).
- Un URI (*Universal Resource Identifier*). Il s'agit du chemin d'accès au fichier statique ou à l'exécutable dynamique qui fait l'objet de la requête. Si le fichier est un exécutable dynamique, il est également possible de transmettre des paramètres correctement encodés à ce programme dans l'URI.
- La version du protocole que le client prend en charge et veut utiliser. Le serveur peut choisir de répondre avec une version plus ancienne du protocole si celle qu'utilise le client n'est pas prise en charge (si cette information est manquante, le client est

supposé utiliser HTTP/0.9, une des premières versions du protocole, maintenant obsolète et que nous n'aborderons pas ici).

Une requête HTTP peut par exemple ressembler à ceci :

```
GET /show_plush_toys.cgi?param1=value&param2=this+is+a+test HTTP/1.1
Host: www.plush-penguins.com
User-Agent: Joe's Own Web Client (UnixWare)
Accept: text/html, text/plain, audio/wav
Accept-Language: pl, en
Connection: close
```

Cette requête demande une ressource intitulée `/show_plush_toys.cgi` au site www.plush-penguins.com. À en juger par l'extension CGI du fichier, il s'agit d'un programme exécuté dynamiquement qui est appelé avec deux paramètres (`param1` et `param2`) indiqués après le point d'interrogation.

La requête du client peut être (et c'est effectivement le cas dans cet exemple) suivie par un certain nombre de champs d'en-tête (un par ligne) qui précisent certains paramètres supplémentaires. Il peut s'agir de l'identification du client (le champ User-Agent, comme indiqué précédemment), de la langue préférée pour le contenu (ici le polonais et l'anglais), en passant par la spécification d'un serveur virtuel auquel le client se réfère (si plusieurs noms de domaine pointent vers une seule adresse IP, cela permet au serveur de déterminer si l'utilisateur est à la recherche de www.squeaky-ducks.com ou de www.plush-penguins.com, qui peuvent être hébergés sur le même système).

Le protocole exige certains de ces en-têtes. L'ensemble des en-têtes requis dépend de la version du serveur, mais la plupart des serveurs sont assez laxistes et ne réagissent pas du tout si certains en-têtes sont omis. Cela mis à part, certains en-têtes précisent certains éléments qui vont au-delà des spécifications du protocole lui-même.

Chaque requête doit se terminer par une ligne vide, pour indiquer la fin des en-têtes client, ce qui, pour la plupart des types de requêtes, signifie que le serveur doit traiter la requête et produire une réponse. Le serveur répond habituellement par un message dont la structure est semblable à celle de la requête et qui commence par un code HTTP en retour et du texte descriptif, comme celui-ci :

```
HTTP/1.0 404 Not Found
Content-Type: text/plain
Server: Uncle Mary's Cookie Recipe Server (Linux and proud of it!)
Date: Mon, 09 Feb 2004 19:45:56 GMT

Le document que vous recherchez est introuvable.
```

Le code ou le message en retour peut signaler que la requête a été accomplie correctement, indiquer au navigateur qu'il doit chercher à un autre endroit ou être un message d'erreur comme "Fichier introuvable" ou "Autorisation refusée". Cette information est suivie par un ensemble d'en-têtes semblable au format accepté pour la requête.

Ces en-têtes décrivent différents paramètres, comme la version du logiciel du serveur, l'emplacement où le navigateur doit ensuite se diriger, le type de contenu du fichier renvoyé, un paramètre utilisé pour différencier les images du texte ou les documents HTML des fichiers binaires, et ainsi de suite. Vient ensuite le contenu, s'il est disponible.

Comme vous pouvez le voir, HTTP est à la base assez simple. Même s'il offre quelques fonctions avancées, la plupart sont soit un peu étranges, soit rarement utilisées (je suppose que vous ne voyez pas le message d'erreur "402 - reformuler votre paiement" tous les jours). Pourtant, il serait naïf de croire que le protocole de base est suffisant pour satisfaire les besoins et les attentes des utilisateurs d'aujourd'hui.

Améliorer HTTP

L'époque où un site Web typique était composé de plusieurs kilo-octets de texte statique et de quelques éléments graphiques est révolue depuis longtemps. Les ordinateurs sont devenus plus puissants, les modems à 300 bauds se trouvent plus facilement dans un musée que dans un foyer et la forme a commencé à supplanter le fond sur le Web. Des centaines de kilo-octets d'images, de pages secondaires et de scripts côté client sont couramment utilisés pour rendre les sites plus attrayants et professionnels, avec plus ou moins de bonheur. Pour de nombreux sites, le contenu multimédia constitue en fait le principal type d'informations présentées, le HTML ne représentant plus qu'un espace pour intégrer des images, de la vidéo, des programmes Java ou des jeux. Le Web en général n'est plus simplement une façon de montrer aux autres ses projets ou ses centres d'intérêt personnels ; ce qui compte est la capacité à commercialiser des produits et à offrir des services moins chers et plus rapidement que jamais. Or le marketing exige que ces produits et ces services soient présentés d'une façon qui attire l'attention.

Les navigateurs Web, les serveurs Web et le HTTP lui-même ont dû s'adapter à cette évolution pour rendre plus facile le déploiement de nouvelles technologies et suivre les nouvelles tendances. Un grand nombre des technologies introduites dans ce processus ont des implications intéressantes sur la sécurité pour un simple mortel et peuvent également nous aider à identifier le client à l'autre bout du réseau de manière transparente. C'est pourquoi nous devons considérer les options et les extensions introduites depuis le jour où le Web est né.

La réduction de la latence : du bricolage

Le problème avec le Web et d'autres protocoles actuels est que les éléments présentés à l'utilisateur sur un seul site multimédia proviennent de sources diverses (voire de domaines totalement différents) et sont ensuite réunis ensemble sur une page. Le texte et les informations de mise en forme des pages Web sont séparés des images et des autres

éléments de grande taille (ce qui est parfait pour les personnes qui ne disposent que d'une bande passante limitée et souhaitent simplement obtenir des informations).

Les clients doivent donc nécessairement effectuer plusieurs requêtes pour procéder au rendu d'une page Web. On pourrait naïvement croire que le meilleur moyen d'y parvenir serait de demander chaque élément l'un après l'autre dans l'ordre, mais c'est loin d'être le cas en réalité, car cela provoque des goulets d'étranglement : pourquoi attendre qu'une page se charge tout simplement parce que le serveur où se trouve la bannière fonctionne lentement ? Ainsi, afin d'améliorer la vitesse de récupération du contenu, le navigateur émet de nombreuses requêtes simultanément.

Et là réside la première faiblesse du protocole HTTP : il n'est pas capable en natif de procéder à des requêtes simultanées mais doit les effectuer l'une après l'autre.

Ce modèle de recherche *séquentielle* (aussi appelé *en série*) pénalise considérablement les performances si l'un des éléments de la page Web doit être téléchargé à partir d'un serveur lent ou depuis une liaison peu rapide ou si cela prend un certain temps au serveur pour préparer et fournir un élément en particulier. Si la recherche séquentielle était la seule option, chaque requête lente empêcherait les requêtes suivantes d'être émises et satisfaites jusqu'à ce que la requête lente soit complétée.

Comme les nouvelles versions de HTTP n'ont pas amélioré cette situation, la plupart des logiciels clients implémentent un bricolage : le navigateur Web ouvre simplement un certain nombre de sessions TCP/IP simultanées et séparées sur un serveur ou un ensemble de serveurs et essaie d'effectuer plusieurs requêtes à la fois. Cette pratique est en fait tout à fait saine lorsque la page demande des ressources à des ordinateurs distincts. En revanche, ce n'est pas une bonne solution lorsque les ressources demandées se trouvent sur un seul système car toutes les requêtes pourraient être effectuées dans une seule session et raisonnablement gérées par le serveur. Voici pourquoi :

- Le serveur n'a aucune chance de connaître quel est l'ordre le mieux approprié pour traiter les requêtes (s'il le pouvait, il fournirait en dernier les objets les plus lourds, ceux qui demandent le plus de temps de traitement ou ceux qui sont tout simplement les moins pertinents). Il est simplement obligé de presque tout faire à la fois, ce qui peut encore retarder inutilement les éléments les plus importants puisque la charge du processeur augmente.
- Si plusieurs ressources de grande taille sont traitées à la fois et que le gestionnaire du système d'exploitation bascule entre les sessions, le résultat peut avoir un impact très négatif sur les performances, car le disque dur doit alternativement rechercher deux fichiers distants de façon répétée.
- Chaque nouvelle poignée de main TCP/IP entraîne généralement une surcharge considérable (bien que cela soit quelque peu atténué par les capacités *keep-alive* dans les nouvelles versions de HTTP). Il est plus efficace d'effectuer toutes les requêtes au sein d'une même connexion.

- L'ouverture d'une nouvelle session et le déclenchement d'un nouveau processus pour satisfaire la requête impliquent une surcharge sur le système d'exploitation et des contraintes sur certains périphériques comme les pare-feu à états. Bien que les serveurs Web modernes tentent de minimiser ce problème en conservant des processus persistants en réserve pour accepter les requêtes au fur et à mesure qu'elles arrivent, le problème est rarement complètement éliminé. Une seule session évite une surcharge inutile et permet au serveur d'allouer seulement les ressources absolument nécessaires pour servir de manière asynchrone les requêtes choisies.
- Enfin et surtout, si le goulet d'étranglement est dû au réseau et non au serveur, les performances peuvent en fait se détériorer car des paquets sont abandonnés puisque la liaison est saturée de données qui arrivent de plusieurs sources simultanément.

Hélas, bonne ou mauvaise, on doit pour le moment se contenter de cette architecture, qui est malgré tout toujours préférable à une recherche séquentielle. On doit tenir compte de son existence et apprendre à en tirer parti.

Comment cette propriété peut-elle bien aider à identifier le logiciel qu'utilise le client ? C'est très simple. La récupération de fichier en parallèle revêt une importance assez évidente pour le fingerprinting du navigateur : il n'y a pas deux algorithmes de récupération simultanés identiques, et il existe de bons moyens de mesurer cela.

Mais, avant de tourner notre attention sur la recherche en parallèle, nous devons examiner deux autres éléments importants concernant la sécurité et la confidentialité sur le Web : les caches et la gestion de l'identité. Même si ces deux sujets ne semblent pas avoir de lien apparents, ils créent un ensemble logique à la fin. Voilà pourquoi nous allons brièvement faire un entracte.

La mise en cache du contenu

La conservation des documents reçus par le serveur dans des caches locaux est une des caractéristiques les plus importantes du développement rapide d'Internet ces dernières années*. Sans cette mise en cache, le coût de fonctionnement aurait été beaucoup plus élevé.

L'augmentation du poids et de la complexité des sites Web exige de plus en plus de bande passante (qui reste généralement assez chère pour les entreprises) ainsi que de meilleurs serveurs pour fournir les données à une vitesse raisonnable.

---

* Toutefois, son importance diminue progressivement : comme de plus en plus de pages Web sont générées dynamiquement et que le backbone Internet devient de plus en plus adulte et se dote de nouvelles capacités, la mise en cache est destinée à perdre de son importance.

Si les performances ne sont pas affectées par les goulets d'étranglement de la bande passante, des solutions comme les sessions simultanées (décrites plus haut) ajoutent des contraintes supplémentaires pour les fournisseurs d'accès. La raison peut sembler surprenante : si une personne disposant d'une liaison assez lente (par modem, par exemple) ouvre quatre sessions à la suite pour récupérer ne serait-ce qu'une page assez simple, quatre connexions et quatre processus ou fils doivent être maintenus sur le serveur, ce qui rend ces ressources inaccessibles pour ceux qui ont des connexions plus performantes.

Enfin, pour aggraver les choses, les sites Web les plus lourds et les plus complexes ne répondent pas toujours aux attentes des utilisateurs. Les utilisateurs actuels ne supportent plus qu'une page Web mette un temps relativement long à se charger. En fait, les études indiquent que l'utilisateur Web n'attend en moyenne pas plus de 10 secondes qu'une page se charge avant d'abandonner[5]. Cela signifie que les entreprises et les fournisseurs d'accès ont besoin de plus de ressources et de meilleures liaisons pour gérer le trafic entrant. En fait, si les choses étaient restées telles qu'elles l'étaient à l'origine, la demande de ressources côté serveur aurait sans doute depuis longtemps excédé la capacité des serveurs à satisfaire cette demande.

Heureusement, les éléments fournis aux internautes sont souvent statiques ou changent rarement, du moins si on compare la fréquence à laquelle ce contenu est modifié à la vitesse à laquelle une ressource est récupérée par les utilisateurs (ceci est particulièrement vrai pour les gros fichiers, comme les images, les vidéos, les documents, les fichiers exécutables, etc.). En rapprochant de l'utilisateur final la mise en cache des données, que ce soit au niveau du fournisseur d'accès Internet ou même sur le navigateur d'extrémité lui-même, on peut considérablement diminuer la bande passante et donc faciliter la gestion du trafic pour les serveurs. En effet, comme plusieurs utilisateurs partagent le même moteur de cache, ils profitent des éléments mis en cache par les précédentes visites. Le FAI bénéficie également de la baisse de consommation de la bande passante, ce qui lui permet de desservir plus de clients sans avoir à investir dans de nouveaux équipements et de nouvelles connexions. Toutefois, HTTP a besoin d'un mécanisme pour que la mémoire cache contienne les bons éléments et qu'elle soit mise à jour. L'auteur d'une page (un être humain ou une machine) doit être en mesure de signaler au moteur de cache quand récupérer une version plus récente d'un document.

Pour implémenter la mise en cache de document, HTTP intègre deux fonctionnalités :

- Une méthode pour indiquer, avec un minimum d'effort, si une partie des données a été modifiée par rapport à la version la plus récente que le moteur de cache détient (le document enregistré lors de la dernière visite).

- Une méthode pour savoir quelles portions des données ne doivent pas être mises en cache, que ce soit pour des raisons de sécurité ou parce que ces données sont générées dynamiquement à chaque fois que la ressource est demandée.

En pratique, cette fonctionnalité est réalisée assez simplement : le serveur renvoie tous les documents pouvant être mis en cache avec la session HTTP, mais en leur ajoutant l'en-tête de protocole Last-Modified. Comme on peut s'y attendre, cet en-tête représente l'idée que le serveur se fait de la date à laquelle ce document a été modifié pour la dernière fois. En revanche, les documents qui ne peuvent pas être mis en cache sont marqués par le serveur avec l'en-tête Pragma: no-cache (Cache-Control: no-cache en HTTP/1.1).

Le navigateur client (ou un moteur de cache intermédiaire géré par le fournisseur d'accès à Internet) est censé mettre en cache une copie de chaque page qui peut l'être en fonction de la présence de l'en-tête approprié, avec les dernières modifications. Il doit conserver la page mise en cache aussi longtemps que possible, jusqu'à ce que la limite du cache configuré par l'utilisateur soit dépassée ou que l'utilisateur purge manuellement le cache, à moins que l'en-tête Expires ne lui indique expressément de la supprimer après une certaine date.

Plus tard, lorsque le site est visité de nouveau, le client considère qu'il dispose d'une version antérieure de la page mise en cache sur le disque et suit une procédure légèrement différente pour y accéder. Tant que le document est dans le cache, le client cherche à obtenir le fichier chaque fois que l'utilisateur revient sur ce site, mais précise l'option d'en-tête If-Modified-Since dans chacune de ses requêtes, en utilisant la valeur <Since> précédente dans l'en-tête Last-Modified. Le serveur compare la valeur Modified-Since avec la date de la dernière modification de la ressource qu'il connaît. Si la ressource n'a pas été modifiée depuis cette date, le message d'erreur HTTP "304 Not Modified" est renvoyé au lieu des données demandées. Par conséquent, le transfert réel du fichier est supprimé, de manière à préserver la bande passante (seules quelques centaines d'octets sont échangés au cours de cette communication). Le client (ou le moteur de cache intermédiaire) est censé utiliser une copie de la ressource mise en cache auparavant plutôt que la télécharger à nouveau.

NOTE

Une approche plus récente repose sur les en-têtes ETag et If-None-Match, qui font partie des fonctionnalités de balisage de HTTP/1.1. Leur fonctionnement est identique mais ils ont pour but de résoudre l'ambiguïté qui entoure l'interprétation de la date de modification des fichiers lorsqu'un fichier est modifié à plusieurs reprises dans un court laps de temps (en dessous de la résolution de l'horloge utilisée pour les données Last-Modified) ou lorsque des fichiers sont restaurés à partir d'une sauvegarde (la date de modification étant alors antérieure à celle de la dernière copie mise en cache), et ainsi de suite.

La gestion des sessions : les cookies

Bien que cela soit apparemment sans lien, il est également important que HTTP soit en mesure de distinguer les sessions et d'effectuer leur suivi à travers les connexions, de stocker les paramètres et les informations d'authentification. Certains sites Web, par exemple, profitent pleinement de leur capacité à s'adapter et à restaurer les préférences personnelles de l'utilisateur chaque fois qu'il visite le site. Bien sûr, l'identité de l'utilisateur peut être établie en l'incitant à saisir un identifiant et un mot de passe chaque fois qu'il affiche une page, ce qui permet ensuite de charger ses paramètres personnels, mais le nombre de personnes disposées à faire ce petit effort supplémentaire pour accéder à la page diminue alors considérablement.

Une façon transparente et persistante de stocker et de récupérer certaines informations depuis la machine client était nécessaire pour assurer un accès personnalisé et invisible aux forums, aux bulletin-boards, aux chats et aux nombreuses autres caractéristiques de la navigation Web pour un si grand nombre de personnes. En revanche, si les administrateurs des serveurs Web reconnaissent et identifient les visiteurs en leur attribuant à chacun une balise, cela signifie que l'utilisateur perd en anonymat ce qu'il gagne en confort d'utilisation. Des entreprises dont la moralité est douteuse peuvent utiliser ce mécanisme pour suivre et établir le profil des utilisateurs, enregistrer leurs préférences d'achats et de navigation, connaître leurs centres d'intérêt, et ainsi de suite. Les moteurs de recherche peuvent facilement mettre en corrélation les requêtes d'un même utilisateur et les fournisseurs de contenu publicitaire et utiliser ces informations pour suivre les internautes, même sans leur permission ou sans que le responsable du site ne le sache*. Ces préoccupations mises à part, il semble toutefois qu'il n'existe aucune autre alternative qui soit suffisamment universelle pour ce mécanisme. Et ainsi naquirent les cookies Web.

Les cookies, définis dans la spécification RFC2109[6], sont de petites portions de texte qui sont émises par un serveur lorsque le client se connecte à celui-ci. Le serveur définit un en-tête Set-Cookie dans sa réponse au visiteur. D'autres paramètres donnent à cette portion de texte une portée limitée à un domaine, à un serveur ou à une ressource spécifique et une durée de vie limitée. Les cookies sont stockés par le logiciel client qui les autorise dans un fichier ou un dossier conteneur spécial et sont automatiquement renvoyés au serveur à l'aide d'un en-tête Cookie chaque fois qu'une connexion à une ressource spécifique est de nouveau établie.

---

* Si une bannière publicitaire ou tout autre élément d'un site Web est placé sur un serveur partagé, comme http://banners.evilcompany.com, l'opérateur de evilcompany.com peut émettre et récupérer des cookies chaque fois qu'une personne visite un site Web légitime qui utilise une de ses bannières. Inutile de dire que la plupart des fournisseurs de bannière utilisent les cookies et suivent les utilisateurs, mais ils le font surtout pour réaliser des études de marché.

Les serveurs peuvent choisir de stocker (ou pas) les paramètres utilisateur définis dans les en-têtes Set-Cookie et se contenter de les lire lors des visites suivantes ; idéalement, c'est uniquement à cela que serviraient les cookies. Malheureusement, les ordinateurs n'ont aucun moyen de dire ce qui est stocké dans un cookie. Un serveur peut choisir d'affecter un identifiant unique à un client en utilisant l'en-tête Set-Cookie, puis le lire en retour pour établir un lien entre l'activité courante de l'utilisateur et ses précédentes actions dans le système.

Beaucoup considèrent que ce mécanisme a de graves répercussions sur la confidentialité. Certains militants haïssent littéralement les cookies, mais l'opposition à cette technologie se réduit de plus en plus de nos jours. Naviguer sur le Web sans jamais accepter de cookies devient de plus en plus difficile, voire impossible sur certains sites qui refusent le trafic des clients qui ne passent pas le contrôle des cookies. Heureusement, de nombreux navigateurs disposent de paramètres permettant d'accepter, de restreindre ou de rejeter les cookies et peuvent même demander si chaque nouveau cookie doit être accepté (bien que cela ne soit pas particulièrement pratique). Il est ainsi possible de protéger de façon raisonnable sa vie privée, ne serait-ce qu'en définissant les sites "autorisés" et les sites de confiance.

Mais contrôlons-nous notre vie privée, alors ?

Le mélange des cookies et du cache

Le respect de la vie privée lors de la navigation sur le Web a longtemps été considéré comme une question brûlante, et ce non sans raison. Beaucoup de gens ne veulent pas que d'autres espionnent leurs préférences et leurs centres d'intérêt, même si leur activité est tout à fait légitime. Pourquoi ? On peut ne pas vouloir qu'une société de publicité sache qu'on est en train de lire un article sur une maladie en particulier puis qu'elle relie cette information à un compte que vous avez sur un bulletin-board professionnel, surtout qu'il n'existe aucun moyen de savoir qui disposera de cette information.

Le contrôle des cookies rend la navigation relativement confortable, tout en écartant les importuns. Mais même la désactivation des cookies n'empêche pas l'information d'être stockée sur un système et d'être renvoyée à plus tard à un serveur. Tous les navigateurs ont longtemps disposé d'une fonctionnalité permettant de stocker et de récupérer des données sur la machine d'une victime, indépendamment de ses paramètres concernant les cookies. Les deux technologies nécessaires, les cookies et les fichiers mis en cache, fonctionnent d'une façon similaire et seuls leurs objectifs diffèrent.

Au cours de l'année 2000, Martin Pool posta une remarque courte mais intéressante[7] accompagnée d'un peu de code sur la liste de diffusion Bugtraq. Il ne voyait aucune différence significative entre les fonctionnalités Set-Cookie et Cookie d'une part et Last-Modified et If-Modified-Since d'autre part, du moins pour les systèmes qui n'utilisent

pas de proxy-cache centralisé et qui stockent les copies des documents déjà consultés sur le disque local (comme c'est le cas pour la plupart des simples mortels que nous sommes). Un administrateur de site Web malveillant peut stocker à peu près n'importe quel message dans l'en-tête Last-Modified renvoyé pour une page que la victime visite (si l'option sanity-check est activée pour cet en-tête, il peut également simplement utiliser une date unique et arbitraire pour identifier ce visiteur en particulier). Le client envoie ensuite l'en-tête If-Modified-Since avec une copie exacte de l'identifiant unique stocké par un opérateur malveillant sur son ordinateur, chaque fois qu'une page est revisitée. Une réponse "304 Not Modified" s'assure que ce "cookie" n'est pas supprimé.

Empêcher l'attaque utilisant les cookies en cache

Utiliser son navigateur pour modifier légèrement les données Last-Modified en retour semble être une bonne façon d'empêcher ce type d'exposition (tout en introduisant certaines inexactitudes de cache), mais ce n'est pas le cas. Une autre variante de cette attaque se fonde sur le stockage des données dans les documents mis en cache plutôt que sur l'utilisation directe des balises : un opérateur malveillant peut préparer une page spéciale pour la victime lorsqu'elle visite un site pour la première fois. La page contient une référence à un nom de fichier unique figurant dans la liste des ressources incorporées (une image, par exemple). Quand un client revient sur cette page, le serveur note la présence de l'en-tête If-Modified-Since, répond avec le message d'erreur 304 et demande donc que l'ancienne copie de la page soit utilisée. L'ancienne page contient une référence de fichier unique qui est ensuite demandée par le serveur, ce qui permet d'établir un lien entre l'adresse IP du client et la session précédente au cours de laquelle le nom de fichier a été renvoyé. Oups !

Naturellement, la durée de vie des "cookies" basés sur le cache est limitée par la taille du cache et les paramètres d'expiration des documents mis en cache selon la configuration de l'utilisateur. Toutefois, ces valeurs sont généralement très généreuses, si bien que l'information stockée dans les métadonnées pour une ressource qui est réexaminée une fois toutes les deux semaines peut durer des années, jusqu'à ce que le cache soit purgé manuellement. Pour les entreprises qui fournissent des composants communs à des centaines ou à des milliers de sites (encore une fois, les bannières sont un bon exemple), cela n'est pas anodin.

La principale différence entre ces cookies mis en cache et les cookies à proprement parler vient non pas des fonctionnalités qu'ils offrent mais plutôt de la facilité avec laquelle on peut contrôler l'exposition mentionnée plus tôt (la mise en cache de données doit aussi servir à d'autres fins et ne peut pas être facilement restreinte. La désactivation partielle ou complète de la mise en cache a un grand impact sur les performances).

Ces étranges contorsions vous montrent comment deux aspects du Web entrent en collision, en annulant effectivement les mesures de sécurité construites autour de l'un d'eux. La pratique montre que les intentions ne suffisent pas toujours, car les attaquants ne sont pas toujours disposés à respecter les règles ni à utiliser la technologie comme on voudrait qu'ils le fassent. Peut-être que désactiver les cookies ne fait pas beaucoup de différence, après tout ?

Mais il est maintenant temps de revenir à l'objet principal de notre discussion.

La découverte des trahisons

Autrement dit, comment détecter les subterfuges et procéder au fingerprinting précis des logiciels client. J'ai indiqué jusqu'à présent qu'il était difficile mais pas impossible de détecter les faux clients et que l'analyse comportementale, un contrôle attentif de la série d'événements produits par les navigateurs en question, était une voie qui mérite d'être explorée.

HTTP est un objet d'étude particulièrement intéressant car, comme nous l'avons vu, une grande partie de l'activité se déroule en parallèle ou presque d'une part et parce que les algorithmes de mise en attente et de traitement des données sont légèrement différents pour chaque client d'autre part. En mesurant le nombre de fichiers téléchargés simultanément, les délais relatifs entre chaque requête dans le temps, l'ordre des requêtes et d'autres petits détails d'une session, il est possible de mesurer les caractéristiques uniques d'un système à un niveau qui est beaucoup plus difficile pour l'utilisateur à manipuler. Ainsi, vous pouvez sans effort distinguer les usurpateurs des citoyens qui respectent les lois.

Pour donner un exemple réel et le plus simple qui soit de cette démarche tout en restant aussi proche que possible des applications réelles, j'ai décidé de voir ce qu'on pourrait apprendre à partir des échantillons de données existants et assez limités dont beaucoup d'entre vous disposent probablement. J'ai donc examiné les journaux standard d'un peu plus de 1 million de requêtes effectuées sur un site Web populaire. Les données utilisées pour cette analyse proviennent d'un journal des accès à un serveur Web Apache. Il s'agit autrement dit des temps d'achèvement des requêtes, des requêtes d'URI, des données transmises au navigateur par l'en-tête User-Agent et d'autres informations de base de ce genre. Le journal est créé à partir d'une page composée d'un ensemble d'images de tailles relativement petites et comparables ainsi que d'un seul document HTML qui appelle tous ces éléments.

Exemple simple d'analyse comportementale

On pourrait considérer comme un problème le fait qu'Apache insère les requêtes dans le journal des accès lorsqu'elles sont terminées plutôt qu'au moment de leur émission, mais c'est en fait très utile, en supposant que l'ensemble des fichiers demandés est relativement homogène. L'ordre de lancement des requêtes suit généralement l'ordre dans lequel les ressources sont référencées dans la page principale, contrairement au temps d'accomplissement, qui est beaucoup plus complexe.

L'ordre d'achèvement dépend du nombre de requêtes, des délais entre chaque requête et d'autres paramètres qui varient légèrement mais sensiblement d'un navigateur à l'autre. Les navigateurs qui gardent toujours une seule connexion ouverte effectuent toujours les requêtes dans un ordre connu, A-B-C-D ; les navigateurs qui ouvrent trois connexions à la fois et émettent les requêtes rapidement sont aussi bien susceptibles de les produire dans l'ordre B-A-C-D que C-B-A-D ou C-A-B-D… Pour ces derniers cas, la mise en attente des requêtes et la gestion de la session compte plus que tout.

Naturellement, nous ne devons pas oublier que la séquence observée dépend également beaucoup de la latence, de la fiabilité et d'autres aléas du réseau. Néanmoins, on peut raisonnablement s'attendre à ce que, pour un grand nombre d'échantillons, ces effets qui ne sont pas spécifiques à chaque navigateur soient en moyenne toujours les mêmes ou affectent les données destinées à tous les clients de la même manière. Et, quand cela se produit, on peut espérer que soient visibles les légères différences entre les navigateurs qui se trouvent sous une interface utilisateur conviviale.

La Figure 14.1 montre une répartition statistique des tentatives de charger la page Web de dix éléments que nous avons mentionnée plus haut par les quatre clients Web les plus populaires. Chaque graphique est divisé en dix segments principaux. Le premier correspond au fichier HTML principal, qui est directement demandé et constitue évidemment le premier élément du site. Les neuf autres segments correspondent aux neuf images référencées sur ce fichier HTML, dans l'ordre où ils sont appelés par le code HTML.

Chacun des segments est ensuite divisé en dix sections sur l'axe X (qui n'est pas indiqué ici pour éviter de surcharger le graphique). La hauteur de la courbe du graphique à la nième section, dans un segment donné, représente la probabilité que le fichier correspondant soit chargé dans le nième ordre.

Pour rendre le graphique plus lisible, les probabilités de distribution sont données en pourcentage entre 1 et 100 (correspondant à un pourcentage, avec toutes les valeurs de moins de 1 % arrondies à l'unité supérieure), et les points se sont connectés à des lignes. Les graphiques sont ensuite reportés sur une échelle logarithmique (log10, logarithme en base 10, avec les principaux guides à 1, 10 et 100) pour rendre les légères différences plus marquées et plus faciles à comparer visuellement.

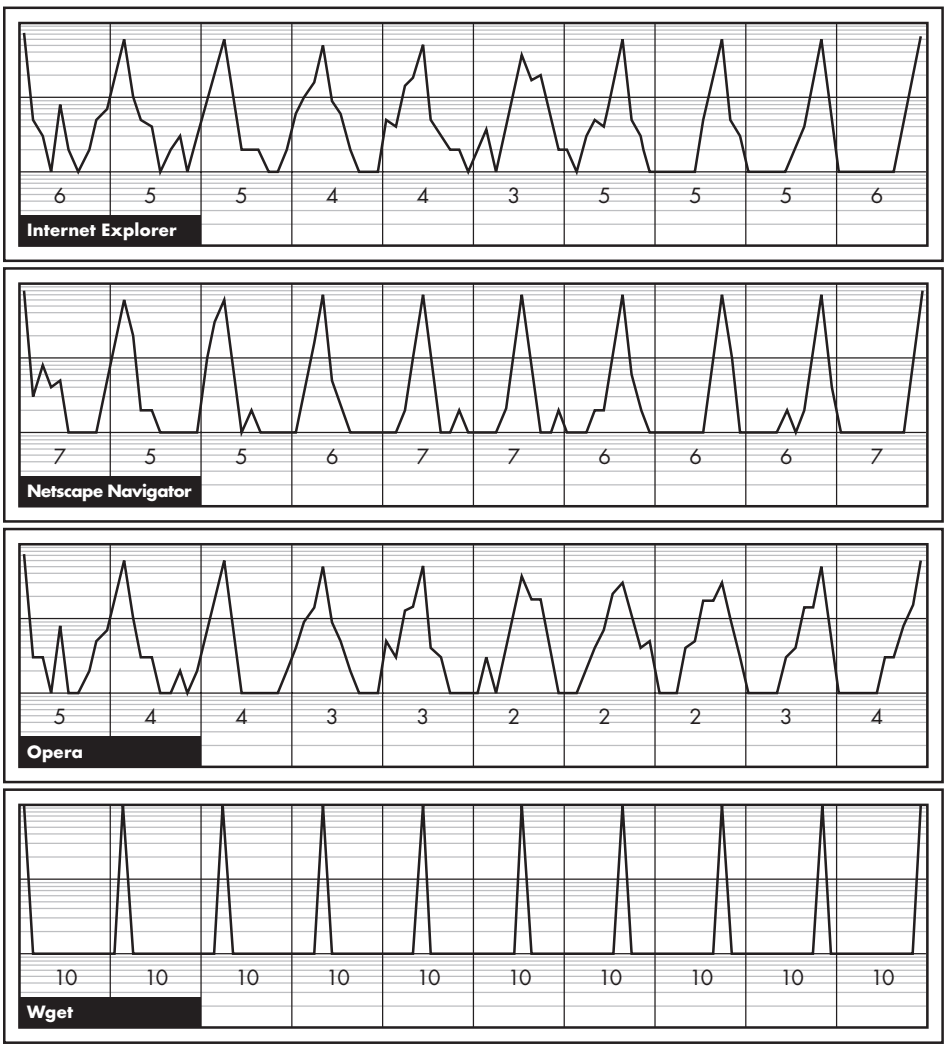


Figure 14.1

Les différences de comportement des clients Web courants.

Idéalement, avec des navigateurs totalement séquentiels et prévisibles, le premier segment ne contiendrait qu'un pic dans la première section (à gauche), le second segment contiendrait seulement un pic dans la deuxième section, et ainsi de suite. Dans la pratique, en revanche, certains navigateurs effectuent plusieurs requêtes simultanément, si bien que cet ordre est plus rarement respecté : le troisième fichier référencé peut

finir par être chargé avant le deuxième ou après le quatrième. Moins un seul pic est prononcé dans chaque segment, plus l'algorithme de récupération du navigateur semble être agressif, autrement dit plus la probabilité que ce fichier ne soit pas chargé dans l'ordre augmente.

Les différences devraient être clairement visibles, même entre les navigateurs qui se basent historiquement sur le même moteur, comme Mozilla et Internet Explorer. Tous les clients semblent respecter l'ordre dans lequel les fichiers sont référencés dans le document principal, si bien qu'à chaque segment le sommet du pic se décale vers la droite par rapport à celui qui le précède. Pourtant, comme vous pouvez le voir, Mozilla est généralement beaucoup moins impatient qu'Internet Explorer et finit le plus souvent par télécharger les fichiers dans l'ordre dans lequel ils ont été demandés. Opera, en revanche, présenté comme le navigateur le plus rapide de la planète, est considérablement moins séquentiel (de nombreux fichiers ont deux ou trois pointes prononcées presque identiques, ce qui semble indiquer qu'un ensemble de requêtes est effectué si rapidement que la séquence d'achèvement est quasi arbitraire et la plus fortement influencée par les fluctuations du réseau). Wget, un gestionnaire de téléchargement Web open source, est en comparaison parfaitement séquentiel (un schéma commun pour les logiciels d'exploration automatiques), utilise une seule connexion et charge tous les fichiers dans le même ordre.

Donner un sens aux graphiques

Les images et les graphiques ne sont guère ou pas du tout utiles pour l'exécution automatisée d'une politique de sécurité ou la détection des abus. Pour quantifier les motifs observés et rendre le fingerprinting un peu plus réaliste, j'ai décidé de donner un meilleur score à un segment (dans une fourchette de 0 à 10) quand il ne contient qu'un seul pic et de lui donner une note plus faible lorsque la distribution est plus arbitraire. Cela pourrait permettre de créer un fingerprinting simple de dix valeurs pour un logiciel en particulier puis, en faisant correspondre l'activité observée à un ensemble de signatures, de déterminer le meilleur.

Pour construire un indicateur qui exprime une qualité relative (linéarité) *Q* du comportement observé dans le segment principal *s*, j'ai utilisé la formule suivante (f_n indique la probabilité que le fichier apparaisse à la position *n* dans la séquence de récupération, exprimée en pourcentage par commodité et pour perturber les puristes) :

$$Q_s = 1{,}42\left(\frac{\sqrt{\sum_{n=1}^{10} f_n^{\,2}}}{10} - 3\right)$$

Cette équation semble effrayante à première vue mais elle est en fait très simple. Je voulais que la formule donne la préférence au cas où ce fichier est le plus souvent chargé lorsqu'il se trouve à une position fixe dans une séquence (c'est-à-dire lorsqu'une valeur *f* est proche de 100 % et que les autres probabilités sont proches de 0 %) par rapport aux situations dans lesquelles toutes les positions ont autant de chances de se produire (toutes les valeurs *f* à 10 %).

Comme la somme de tous les éléments de *f* est fixe (100 %), le moyen le plus simple d'y parvenir est d'utiliser une somme de carrés : pour toute séquence de chiffres non nulle, la somme des carrés de ces nombres est toujours inférieure au carré de la somme. Les résultats les plus hauts et les plus bas sont les suivants :

$$10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 = 1\,000$$

$$100^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 = 10\,000$$

Le reste de l'équation, outre la somme principale, est utilisé uniquement pour représenter les résultats avec une échelle raisonnable de 0 à 10 (quand ils sont arrondis).

Les résultats du calcul de cette mesure pour chaque segment du trafic observé pour chaque navigateur sont superposés à la Figure 14-1, en tant que valeur numérique décrivant tous les segments du graphique. Comme prévu, le score de Wget est parfait à chaque segment. Les scores des autres navigateurs confirment les observations visuelles précédentes et les rendent plus tangibles. Bien que les moteurs d'Internet Explorer et de Mozilla/Netscape semblent avoir des graphiques à peu près similaires, on peut remarquer de grandes différences de charge pour les éléments 4 à 6 et dans une moindre mesure dans l'ensemble de la séquence de récupération. Opera se distingue clairement des autres avec des scores constamment plus bas pour chaque segment.

En conséquence, cet outil d'analyse assez simple permet d'obtenir un cadre pour élaborer une méthode pratique d'identification des navigateurs et pour détecter des supercheries dans un échantillon statistiquement significatif du trafic HTTP de l'utilisateur. On peut améliorer ce modèle en analysant les autres éléments qui se chargent automatiquement, comme les scripts, les feuilles de style HTML, les images réactives, les cadres et d'autres fichiers qui montrent encore plus les différences entre chaque navigateur. Le père Noël devrait pouvoir préparer la liste des utilisateurs qui n'ont pas été sages cette année.

Au-delà du moteur...

J'espère simplement montrer à quel point il est facile de détecter les caractéristiques cachées d'une application inconnue en observant son comportement, sans émettre d'hypothèse particulière ni disséquer les entrailles de ce programme. Les chiffres exacts ci-dessus ne sont pas directement utilisables sur un autre site Web que celui que j'ai utilisé.

Je vous encourage donc à faire vos devoirs, si vous pensez que cette technique peut vous être utile. Une fois que vous disposez du profil d'un site ou d'un ensemble de sites, vous pouvez utiliser les données pour reconnaître efficacement les systèmes en fonction de leur activité dans le temps.

Inutile de dire que la méthode que j'ai utilisée ici est une approche (peut-être trop) simpliste de l'analyse comportementale et se fonde peut-être sur le plus simple des scénarios possibles ; je vous la fournis pour vous encourager et vous inciter à effectuer de plus amples recherches. Vous pouvez facilement utiliser le processus de rendu du contenu des cadres, des tableaux et des autres conteneurs visuels ou le processus de récupération et de rendu de certains types de fichiers afin de connaître le navigateur utilisé, même sans effectuer de correspondance statistique des performances (certains aspects très spécifiques de l'activité des navigateurs présentent des différences beaucoup plus flagrantes). Vous pouvez également vous tourner vers l'analyse des différences dans le temps.

Tenez également compte de la chose suivante : vous pouvez réfléchir plus en profondeur sur l'analyse comportementale et l'utiliser pour différencier les machines des êtres humains, voire pour identifier les utilisateurs plutôt que pour distinguer un moteur de rendu d'un autre. Comme indiqué au Chapitre 8, les modes d'utilisation du clavier sont souvent tellement caractéristiques d'un individu qu'il est possible de les utiliser pour une reconnaissance biométrique. De même, les études semblent indiquer que nous pouvons utiliser la façon dont les utilisateurs cliquent sur les liens, effectuent des choix, lisent les informations et ainsi de suite pour indiquer de qui ou de quoi proviennent un ensemble de requêtes[8]. Bien que plus proche de la spéculation scientifique que de l'étude des faits, il s'agit là d'un merveilleux champ de recherches.

… et au-delà de l'identification

Les applications d'analyse de l'activité et du comportement du navigateur vont au-delà de la détection du logiciel de navigation – en fait, certains relèvent du domaine du respect de la vie privée et de l'anonymat.

Un intéressant travail de recherche publié en 2000 par Edward Felten et Michael Schneider[9] apporte une contribution fascinante aux applications possibles de cette technique, une capacité qui est étroitement liée aux mécanismes de mise en cache déployés dans les moteurs actuels. Tous les éléments examinés jusqu'ici se rejoignent enfin.

Leur étude repose sur le principe suivant : en insérant une référence à un fichier sur un site particulier puis en mesurant le temps que le navigateur met à télécharger ce fichier, il est possible de dire si l'utilisateur a visité un site particulier les jours précédents. Assez simple.

Je vous épargne (pour cette fois) une longue digression dans le monde de la théorie, des prévisions et des spéculations et vous propose plutôt un exemple proche d'une situation réelle. Supposons que je sois le responsable du site www.rogue-servers.com. Pour une raison ou pour une autre, j'ai décidé que ma page principale fera référence à une image (un logo, par exemple) tirée du site www.kinky-kittens.com*. Je rends cet élément visuel difficile à trouver ou je réduis sa taille de façon qu'il ne soit pas visible mais qu'il soit quand même chargé par le navigateur.

Un utilisateur sans méfiance visite mon site. S'il n'a jamais visité www.kinky-kittens.com, il lui faut un certain temps pour télécharger l'image que j'ai référencée. En revanche, s'il s'agit d'un visiteur récurrent de ce site, l'image est déjà présente dans son cache, si bien qu'elle est récupérée presque instantanément.

Comme la référence à la ressource de www.kinky-kittens.com est précédée et suivie de requêtes pour les autres éléments visuels que j'héberge sur mon site, il est possible en déployant des heuristiques dans le temps de mesurer avec fiabilité si l'ensemble du logo a été récupéré ou s'il était déjà dans le cache. Tout cela suffit à savoir si un nouveau visiteur de ma page est en fait un habitué d'un site Web en particulier (ou d'une section particulière d'un site Web), ce qui représente un envahissement brutal de sa vie privée. Bien que ce scénario n'ait que peu de chances d'être utilisé pour déployer des routines d'espionnage à grande échelle (principalement parce que cela laisse une preuve claire qui pourrait être remarquée par l'opérateur du serveur sur lequel se trouvent les utilisateurs qu'on veut espionner), des attaques ciblées pourraient être tout à fait efficaces.

Finalement, toutes les pièces du puzzle s'emboîtent, même si elles le font peut-être d'une façon assez lâche. Les utilisateurs, les programmes et les habitudes peuvent être facilement révélés en abusant soigneusement des caractéristiques récentes d'un protocole Internet populaire. Ce qui n'est pas forcément rassurant pour les visiteurs de www.kinky-kittens.com.

---

* *NdT :* Le site www.kinky-kittens.com est un site pornographique…

Prévention

Rendre totalement anonyme sa navigation sur le Web semble être un combat perdu d'avance. Bien que certaines pratiques visant à améliorer la protection de la vie privée et l'anonymat des internautes soient généralement acceptées, ces fonctionnalités peuvent facilement être contournées par un site Web malveillant.

Le problème est, malheureusement, trop grave pour le négliger. Qu'une entité à laquelle nous avons décidé de faire confiance (comme un FAI) connaisse notre activité, d'accord, mais il n'en va pas de même lorsque des parties avec lesquelles nous ferions mieux de ne pas traiter recueillent systématiquement des informations sensibles sur notre profil et les revendent probablement à d'autres dans le cadre de leurs activités commerciales. Cela suffit à inquiéter même ceux qui ne portent pas de sous-vêtements en aluminium tous les jours.

Par ailleurs, la relative difficulté à rester totalement anonyme ou à apparaître totalement inoffensif est importante dans des environnements où le trafic HTTP doit être autorisé mais où les utilisateurs doivent être protégés et surveillés, sans porter atteinte à leur vie privée au-delà de ce qui est strictement nécessaire. Dans les réseaux d'entreprise, aussi bien les utilisateurs que les administrateurs système apprécient vraiment de pouvoir repérer les systèmes en infraction sans avoir à inspecter manuellement les données.

Matière à réflexion

Aucun élément HTTP n'est en lui-même mal conçu, cassé ou ne présente aucune garantie. Cependant, lorsqu'on les additionne, de nombreuses fonctionnalités de sécurité et de protection de la vie privée semblent s'annuler et laisser l'utilisateur très vulnérable aux indiscrétions. Malheureusement, nous ne pouvons pas faire grand-chose contre cela sans tout reprendre à zéro. Et, dans ce cas, il n'y a aucune garantie que les résultats obtenus fonctionneraient aussi bien ou fourniraient autant de protection de la vie privée que les clients HTTP, HTML et WWW ne le font actuellement.

15

Les avantages d'être une victime

Dans lequel nous concluons que faire preuve d'optimisme peut aider à traquer l'attaquant.

J'ai abordé différents problèmes qui peuvent avoir un impact significatif sur toutes les communications quotidiennes lorsqu'ils sont cumulés et présenter des risques assez embarrassants. Vous avez vu comment les autres peuvent exploiter le réseau pour vous voler des informations ou pour en savoir plus que vous ne pensiez ou ne leur permettriez, mais aussi comment utiliser ces techniques pour recueillir plus d'informations sur votre propre réseau d'entreprise ou domestique et sur les attaquants qui les ciblent. J'espère avoir présenté un aperçu à la fois utile sur la façon dont ces problèmes sont nés et sur celle de les éviter quand cela est possible. J'ai essayé de montrer que la sécurité et la confidentialité ont des implications sur chaque activité et qu'il ne suffit pas de prendre des décisions de conception correctes, d'installer les bons logiciels ni d'établir et de faire respecter les politiques appropriées pour les éliminer totalement. La divulgation d'informations ne peut tout simplement pas être totalement supprimée. Notre seul espoir est d'avoir suffisamment d'informations et de connaissances sur les fuites potentielles ou les scénarios d'attaque pour en atténuer autant que possible les effets les plus significatifs dans une application en particulier.

Dans cette troisième partie de l'ouvrage, j'ai mis l'accent sur les menaces qui se cachent dans les réseaux de grande taille. Bien que cette partie soit la plus longue et n'arrive que maintenant à sa conclusion, elle est très loin de présenter de façon exhaustive tous les problèmes qui peuvent se poser dans un réseau ouvert. En fait, il serait très difficile et en grande partie inutile d'aborder toutes les variantes des problèmes. C'est pourquoi j'ai choisi de ne couvrir que les aspects les plus complexes, difficiles ou fascinants des communications d'hôte à hôte. J'ai mis l'accent sur la découverte des scénarios d'attaque sur les différentes couches de protocole et les différents niveaux d'abstraction, plutôt qu'énumérer les concepts et les vecteurs d'attaque, ressasser de vieilles idées et ne rien apporter de nouveau au sujet. J'espère que les informations fournies jusque-là vous aideront et vous encourageront à trouver d'autres exemples de ces problèmes dans d'autres aspects de la mise en réseau et de l'informatique – voire peut-être au-delà.

Dans la prochaine partie de l'ouvrage, nous effectuerons un important changement de paradigme car nous étudierons comment l'observation attentive de l'ensemble du réseau, par opposition à l'observation de systèmes uniques, peut être utilisée pour nous défendre ou pour attaquer. Mais, avant cela, penchons-nous sur une partie inhabituelle de la surveillance du réseau : le contre-espionnage passif (en apprendre plus sur l'attaquant ou ses objectifs en analysant ses actions). Les données collectées de cette façon peuvent fournir un ensemble de pistes très intéressantes et permettre d'identifier plus facilement les intentions d'une personne malveillante, ses outils ou encore l'attaquant lui-même. Établir le profil d'un attaquant, essayer de lire ses pensées et peut-être même jouer au chat et à la souris avec lui constituent souvent une expérience passionnante en soi.

Connaître les mesures de l'attaquant

Comme prévu, on peut acquérir une bonne partie des informations sur une partie attaquante en utilisant simplement certaines des mesures courantes du trafic TCP/IP que nous avons vues précédemment (comme le fingerprinting passif du système d'exploitation). On peut ainsi identifier l'utilitaire utilisé pour effectuer le scan des ports, par exemple.

De même, on peut aussi appliquer l'analyse comportementale aux caractéristiques du comportement de l'attaquant, comme les retards et l'ordre des requêtes (l'ordre dans lequel les ports sont scannés et à quelle vitesse, par exemple). On peut utiliser l'analyse comportementale pour effectuer le suivi des programmes, voire pour connaître les caractéristiques d'un attaquant (comme sa maîtrise de l'ordinateur) lors d'une effraction effectuée manuellement ou lors de tentatives de découverte non autorisées.

Une méthode particulièrement intéressante que nous pouvons déployer pour identifier l'utilitaire que l'attaquant utilise pour scanner notre réseau s'appuie sur l'application de

l'une des méthodes évoquées au Chapitre 9, le fingerprinting de la séquence des ports, pour une toute nouvelle tâche. Cette méthode repose sur le fait qu'une majorité des scanners utilisés aujourd'hui explorent les réseaux et les systèmes de façon séquentielle (depuis le port ou l'adresse ayant le plus petit numéro vers le port ou l'adresse ayant le numéro le plus élevé) ou accèdent aux ressources dans un ordre aléatoire. Cette dernière approche est la plus souvent utilisée et considérée comme la meilleure car elle peut équilibrer les charges et rendre le scan légèrement plus difficile à détecter. Mais l'utilisation d'un ordre aléatoire peut bizarrement se retourner contre l'attaquant de différentes façons.

Le problème se pose car les auteurs des utilitaires de balayage des ports ne considèrent pas que ces applications soient très importantes ni qu'elles nécessitent un haut niveau de sécurité. La façon la plus courante et la plus simple pour implémenter un générateur de nombres pseudo-aléatoires dans des programmes dont la sortie ne nécessite pas d'être sécurisée par chiffrement consiste à utiliser les langages standard ou intégrés du système. La norme ISO[1] pour le langage de programmation le plus répandu dans le monde, à savoir le langage C, suggère d'utiliser un simple algorithme linéaire congruent pour implémenter une bibliothèque standard C pour le générateur de nombres pseudo-aléatoires (vu au Chapitre 1). Selon cette norme, la recette pour créer et utiliser le générateur est la suivante :

1. Le générateur doit être initialisé avec une valeur initiale 32 bits (S_0), en faisant appel à la fonction de la bibliothèque standard `srand()`. Dans le cas contraire, le générateur utilisera un germe défini par défaut et produira des séquences identiques dans tous les cas.
2. Dans chaque appel à `rand()`, la fonction principale invoquée à plusieurs reprises pour obtenir les suites de nombres pseudo-aléatoires qui seront utilisés dans les applications de l'utilisateur, le germe S est recalculé comme suit :
   $S_{t+1} = S_t * 1103515245 + 12345$. Le résultat est tronqué à 32 bits (modulo 4294967296).
3. La valeur en retour de chaque appel `rand()` est le mot plus important de S_{t+1}, modulo 32768. Dans une variante 32 bits, l'un des algorithmes les plus couramment utilisés sur les ordinateurs actuels, la procédure prévue à cette étape et à l'étape précédente est répétée plusieurs fois pour calculer les portions de bits suivantes de la valeur obtenue comme résultat.

Comme mentionné au Chapitre 1, tous les générateurs linéaires congruents, y compris celui décrit ici, sont sensibles à la méthodologie de cryptanalyse proposée par H. Krawczyk dans les années 1990. En observant les sorties de quelques suites de séquences (qui se suivent ou sont ordonnées différemment), il est possible de reconstituer l'état interne du générateur et de prédire ainsi toutes ses sorties précédentes et futures.

Naturellement, la conséquence immédiate (la victime peut savoir à partir des tentatives précédentes dans quel ordre l'attaquant va essayer de cibler d'autres ressources sur

la machine ou le réseau) n'est pas particulièrement excitante ni n'a de grande valeur en elle-même. Néanmoins, cette possibilité a deux répercussions importantes dans le contexte des tentatives de découvertes du réseau :

- Il est possible de connaître la valeur de S_0. Si on sait ou si on peut estimer quand le générateur a commencé à fonctionner (ou du moins quelles propriétés générales le germe initial devrait avoir), on peut alors reconstruire la valeur utilisée pour initialiser le générateur. Comme S_0 est la seule entrée de l'algorithme, des valeurs de germes identiques doivent avoir un comportement identique. On peut donc suivre la trace des germes en observant la sortie du PRNG.
- Il est possible de connaître les incréments *t*. Une fois qu'on a reconstruit l'état du générateur, on peut savoir combien de valeurs aléatoires ont été demandées par le scanner en utilisant `rand()` entre deux appels que le scanner a effectués pour obtenir les valeurs des paquets (les numéros de port ou les adresses des hôtes).

L'importance de la première conséquence, la capacité à reconstruire la valeur utilisée pour initialiser le générateur, peut ne pas être immédiatement apparente. Mais on doit également tenir compte d'une autre pièce du puzzle. Pour initialiser un générateur de nombres aléatoires, on utilise couramment une valeur 32 bits qui change suffisamment souvent pour éviter que le PRNG ne se comporte trop souvent à l'identique. Pour cela, on utilise souvent le compteur de temps du système, en le combinant parfois avec un autre nombre de petite taille comme l'identifiant de processus courant (PID), afin de réduire le risque que deux programmes qui s'exécutent dans un court intervalle de temps produisent des résultats similaires.

En utilisant ces connaissances à la valeur S_0 calculée, la victime peut découvrir l'heure du système de l'attaquant (heure GMT ou locale, selon les réglages du système d'exploitation et le type de scanner). En connaissant l'heure locale du système, l'observateur a un indice sur l'origine et l'identité de l'attaquant d'une manière très simple. S'il usurpe des paquets provenant de plusieurs sources pour créer une certaine confusion, on peut heureusement exclure les sources où S_0 indique un fuseau horaire qui ne correspond pas à la région géographique à laquelle appartient l'adresse source. Si on compare par exemple l'heure estimée du système de l'attaquant avec l'heure du méridien de Greenwich et qu'on découvre que l'heure de l'attaquant est égale à GMT-5 heures, on peut alors conclure qu'il se trouve sur la côte est des États-Unis et non en Chine. En comparant cette estimation de la zone horaire avec les enregistrements de plusieurs blocs d'adresses IP, on peut savoir malgré tous les "leurres" que la véritable identité de l'attaquant a plus de chances d'être dans les paquets provenant d'un FAI de Boston que dans ceux d'un FAI situé à Pékin.

En outre, une fois connue l'heure locale de l'attaquant, on peut le suivre en mesurant l'écart entre l'horloge de son système et le temps réel (et, à long terme, à quel rythme il

s'en écarte). Comme les horloges des ordinateurs ne sont généralement pas particulièrement précises et ont tendance à retarder ou à avancer un peu quand elles ne sont pas régulièrement synchronisées avec une source externe (cet écart peut s'élever à plusieurs minutes par jour dans certains cas), cela peut représenter un bon moyen pour mettre en corrélation les attaques menées par la même personne. D'une machine à l'autre, le décalage dans le temps est systématiquement différent et évolue différemment.

Enfin, lorsque le PID et l'heure du système sont utilisés dans le cadre de l'initialisation des germes et qu'on dispose d'une évaluation de l'heure du système de l'attaquant, le PID peut permettre de connaître approximativement l'uptime du système ou le nombre de tâches exécutées entre deux scans. Comme chaque nouveau processus sur une machine se voit attribuer un nombre PID plus élevé, cette dépendance est assez simple*.

En reconstruisant l'état du PRNG, on peut aussi voir combien de nombres aléatoires ont été générés entre la création de deux paquets reçus par le destinataire. Quand un seul système est scanné, il ne doit pas y avoir de trous ou d'écarts marginaux dus à des problèmes réseau. En revanche, lorsque plus d'un système est scanné, ces trous (causés par les paquets envoyés à des cibles différentes) peuvent être facilement détectés. Et cette détection permet de déterminer combien de systèmes sont pris pour cible en même temps.

En outre, lorsque le logiciel de scan génère de faux paquets leurres qui semblent provenir d'hôtes aléatoires, il est possible d'éliminer les adresses usurpées qui ont été créées à l'aide du PRNG (elles correspondent à sa sortie possible), de savoir laquelle ne correspond pas et donc doit être réelle. On pointe ainsi de façon concluante vers le véritable auteur d'une attaque. Par exemple si les données du PRNG reconstruit montrent que le trafic provient d'adresses comme :

198.187.190.55 (représentation décimale : 3334192695) ;

195.117.3.59 (représentation décimale : 3279225659) ;

207.46.245.214 (représentation décimale : 3475961302).

On peut alors savoir que 3334192695 et 3475961302 correspondent à l'une des premières sorties que produirait un générateur initialisé avec le germe S_0. Comme 3279225659 ne semble pas correspondre à l'un des premiers résultats produits par un PRNG reconstruit, il s'agit donc probablement d'une adresse réelle.

Ces informations peuvent être utilisées pour connaître les intentions d'un attaquant et les logiciels qu'il utilise. On peut s'en servir pour suivre le système sur lequel il travaille, en mettant ces informations en corrélation avec d'autres données afin de déterminer sa véritable identité et sa situation géographique, voire, parfois, savoir comment il utilise son ordinateur à mesure que le scan des ports progresse.

---

* Bien que certains systèmes offrent des options de randomisation du PID dans le but de rendre plus difficiles certains types d'attaques locales sans liens entre elles.

NOTE

NMAP, en réponse aux problèmes de divulgation de l'uptime et de l'historique des scans examinés plus haut, tente d'utiliser les mécanismes sécurisés RNG (comme /dev/random, décrit au Chapitre 1) pour générer des nombres aléatoires, au lieu de compter sur les outils de la bibliothèque standard du langage C. Cependant, cette méthode n'est pas disponible sur de nombreux systèmes d'exploitation (comme Windows) et les autres scanners n'ont pas pris de mesures similaires pour défendre l'attaquant..

Se protéger : observer les observations

Internet est devenu un immense champ de bataille au cours des dix dernières années. Dès leur connexion, les machines sont instantanément l'objet d'attaques par des sondes automatiques, des vers et d'autres types d'informations qui sollicitent leur dispositif de sécurité. Les méthodes traditionnelles de détection et de prévention d'intrusion (à la mode actuellement) utilisent des utilitaires d'analyse du trafic spécialement conçus pour identifier et arrêter les attaques et avertir l'administrateur lorsque des découvertes du réseau sont réalisées. Dans des environnements hétérogènes ou tout simplement assez complexes, ces techniques produisent plus de bruit et de résultats faussement positifs qu'il est possible d'en traiter.

Dans certains cas, toutefois, la possibilité d'observer les attaques et les réponses qu'elles déclenchent est un excellent moyen pour l'administrateur d'en savoir plus sur les problèmes du réseau et sur les attaques (même si ces incidents ne méritent le plus souvent pas d'être notés). Dans certains réseaux, il est difficile d'initier ou trop ennuyeux de procéder à la découverte active et à la numérisation des actifs destinés à garantir le respect des politiques et la configuration du système (en raison de la politique de réglementation, de la lenteur des délais d'exécution, des tickets de maintenance du réseau rarement ouverts, etc.). Dans ce cas, la capacité à observer et à savoir ce que les attaquants voient peut être un précieux substitut à la reconnaissance initiée en local.

En outre, la découverte active réalisée périodiquement peut ne pas être suffisamment rapide pour répondre à certaines menaces ; il peut donc être utile d'apprendre que quelque chose a brusquement mal tourné en observant simplement les résultats que les autres obtiennent. Bien entendu, c'est là une arme à double tranchant, un attaquant qui a compromis ou prévoit de compromettre un réseau mais qui souhaite rester prudent et planifie chaque étape à l'avance peut observer le trafic généré par la découverte d'autres tentatives pour en savoir plus sur un système en particulier.

Profiter des connaissances acquises par un attaquant n'est simple qu'en théorie ; il n'est pas facile de mettre en corrélation et de traiter les résultats, en particulier lorsque l'analyse porte sur des environnements vastes ou lorsqu'elle se fonde uniquement sur

des informations partielles obtenues à partir de différentes tentatives d'attaques effectuées depuis plusieurs endroits. Lentement, des outils qui facilitent l'établissement de la cartographie du réseau et du système en utilisant "le balayage passif" font cependant leur apparition – DISCO^2, de Preston Wood, étant un excellent exemple.

Matière à réflexion

Je trouve étrange que les techniques décrites dans le présent chapitre ne soient pas plus étudiées ni décrites dans des livres blancs ou que des utilitaires ne soient pas disponibles sur ce sujet. Vu l'engouement pour le suivi des attaques suscité par l'étude du pot de miel de Lance Spitzner (et que seuls des produits comme les systèmes de détection des intrusions alimentent), on pourrait s'attendre à ce que moins d'efforts soient consacrés à identifier les attaques (qui ne sont généralement pas particulièrement excitantes en elles-mêmes et qui utilisent généralement des vecteurs et des failles bien documentés) au profit de tentatives qui visent à connaître les intentions et les origines d'une attaque et à établir une corrélation entre des événements qui n'ont aucun sens lorsqu'ils sont isolés mais qui peuvent signaler un problème lorsqu'ils sont combinés.

Je ne peux que montrer la partie émergée d'un iceberg, mais il va sans dire que cela pourrait être un domaine des plus passionnants à étudier.

Et, maintenant, quelque chose de complètement différent…

Partie IV

Vision d’ensemble

Notre service juridique nous a conseillé de ne pas baptiser cette partie : "le réseau est l’ordinateur".

16

Le calcul parasitaire, ou comment l'union fait la force

Dans lequel se confirme que mieux vaut avoir une armée de serviteurs que faire le travail soi-même.

J'espère que vous avez apprécié le parcours jusqu'ici. J'ai abordé un certain nombre de problèmes qui affectent la sécurité et la confidentialité des informations depuis leur saisie au clavier jusqu'à leur destination finale à des centaines ou à des milliers de kilomètres de distance. Mais il est encore trop tôt pour s'arrêter là, car quelque chose manque encore, quelque chose de bien plus grand que ce que nous avons vu jusqu'à présent. La matière noire.

Notre histoire jusqu'à présent présente un défaut assez évident : les communications ne se produisent pas dans le vide. Bien que le processus d'échange des données se limite généralement à deux systèmes et à une dizaine d'intermédiaires, il est impossible d'ignorer le contexte général. Les propriétés de l'environnement peuvent influencer en profondeur la réalité d'une communication entre deux terminaux. Nous ne pouvons pas ignorer l'importance des systèmes qui ne sont pas directement impliqués dans les communications ni l'importance de tous les événements minuscules, apparemment isolés, que les données rencontrent en cours de chemin. Il peut être fatal de se concentrer

uniquement sur ce qui semble relever d'une application spécifique ou d'un cas particulier, comme j'espère que ce livre vous l'a montré jusqu'ici.

Plutôt que tomber dans ce piège, j'ai choisi de tout englober. Ainsi, la quatrième et dernière partie de cet ouvrage se concentre exclusivement sur la sécurité du travail en réseau dans son ensemble et examine Internet comme un écosystème et non comme un ensemble de systèmes accomplissant chacun des tâches spécifiques. Nous rendons hommage à la matière apparemment inerte qui relie toutes choses.

Cette partie de l'ouvrage commence par l'analyse d'un concept qui semble être le moyen le plus approprié pour effectuer la transition. Pour beaucoup de geeks informatiques, ce concept, appelé calcul parasitaire, a révolutionné notre façon de penser Internet.

L'utilisation des processeurs distants

Tout commence par un modeste document de recherche publié sous forme d'une lettre dans le journal *Nature* par Albert Laszlo Barabasz, Vincent W. Freeh, Hawoong Jeong et Jay B. Brochman en 2001[1] qui aurait très bien pu passer inaperçu. À première vue, cette lettre ne semblait mériter qu'on lui accorde une grande attention. En fait, la thèse émise semblait même apparemment dérisoire. Ses auteurs indiquaient que du trafic pouvait être créé à l'intérieur de protocoles réseau bien établis comme TCP/IP pour poser (comme un message) un problème arithmétique simple à un ordinateur distant. Le système distant résoudrait alors involontairement le problème tout en analysant le message et en préparant une réponse. Mais pourquoi perdre du temps à poser des énigmes à des machines dépourvues d'émotion ? Qu'aurions-nous à y gagner ? Ne serait-il pas aussi amusant de les résoudre soi-même ? Bien entendu, la réponse est très intéressante.

Premièrement, la résolution de puzzles avec un ordinateur n'est pas si simple : la cryptographie actuelle se fonde en majorité sur la relative difficulté à résoudre une série de problèmes prétendument non déterministes polynomiaux* (NP). Les problèmes NP-complet semblent prendre plaisir à faire échouer chaque attaque sur le code au moment le moins

---

* Selon la théorie de la complexité, un problème polynomial peut être résolu par une machine de Turing dans un temps polynomial proportionnel à la longueur d'entrée (nombre ou taille des variables pour lesquelles la réponse doit être trouvée). Cela signifie que le temps nécessaire pour résoudre un problème polynomial correspond directement à la longueur en entrée élevée à une puissance constante, qui peut être égale à zéro (ce qui fait que le temps ne dépend pas du tout de la longueur en entrée, comme pour les tests de parité). Un problème non déterministe polynomial (NP) n'a pas de solution connue de cette nature et peut exiger beaucoup plus de temps pour être résolu à mesure que la longueur d'entrée augmente et présente par exemple une dépendance exponentielle. Certains problèmes NP, connus sous le nom de NP complets, n'ont aucune solution en temps polynomial. Les problèmes NP sont généralement considérés comme "difficiles" pour des entrées complexes, tandis que les problèmes déterministes sont moins coûteux à résoudre.

opportun. L'inventeur d'une manière efficace de les résoudre, que ce soit grâce à une énorme puissance de calcul, à des algorithmes astucieux ou aux deux, deviendrait quasiment le maître du monde. Il y a donc une bonne raison de chercher à le faire, mais comment y parvenir ?

Cette étude proposait une méthode tout à fait inédite. Tout d'abord, ce document déclarait que de nombreux problèmes NP en mathématiques peuvent être facilement exprimés en termes d'équations SAT (équations booléennes ayant une solution). Les équations SAT représentent ces problèmes sous la forme d'opérations booléennes logiques, en créant une séquence de paramètres et de variables (une formule booléenne). Voici un exemple classique de formule SAT :

$P = (x_1 \text{ XOR } x_2) \text{ AND } (\sim x_2 \text{ AND } x_3)$

Ici, *P* est la formule elle-même (le problème) tandis que x_1, x_2 et x_3 sont les entrées binaires ou les paramètres.

Bien qu'il existe 2^3 combinaisons possibles pour les valeurs x_1, x_2 et x_3, seule l'une d'elles rend *P* vrai : $x_1 = 1$, $x_2 = 0$ et $x_3 = 1$. Par conséquent, seul ce triplet est une solution de *P*. Trouver la solution des problèmes SAT revient donc à déterminer un ensemble de valeurs pour toutes les variables de l'équation, afin que l'ensemble de la formule qui intègre ces variables ait une valeur logique vraie. Bien que les problèmes SAT simples comme celui de notre exemple soient faciles à résoudre uniquement par tâtonnements, les cas plus complexes comprenant de multiples variables sont effectivement NP-complets. Par conséquent, d'autres problèmes NP peuvent être réduits à des problèmes SAT dans un temps polynomial (ou acceptable).

C'est là que réside le problème. Nous pouvons formuler un problème NP difficile avec une équation SAT, mais cela ne nous apporte pas grand-chose de plus. À ce jour, quand il s'agit d'une équation complexe, même les meilleurs algorithmes de résolution SAT connus ne sont pas beaucoup plus efficaces que la recherche par force brute dans laquelle toutes les possibilités sont essayées et la valeur de la formule, évaluée à chaque possibilité. Cela signifie que, si nous avons un problème SAT et suffisamment de puissance de calcul pour envisager de la résoudre, il n'est pas aussi dément que cela de chercher une solution en utilisant la force brute. Une méthode plus sophistiquée n'apporterait pas grand-chose de plus et puis, de toute façon, nous n'avons pas grand-chose à perdre à essayer.

Et voici le lien entre les problèmes SAT et le protocole TCP/IP. Le constat de base des chercheurs est assez évident (ou devrait l'être, si vous souscrivez au journal *Nature*) : l'algorithme de la somme de contrôle de TCP (ou IP) décrit au Chapitre 9, même s'il a en principe été conçu dans un tout autre but que pour résoudre des équations, n'est rien d'autre qu'un ensemble d'opérations booléennes réalisées à la suite sur les bits du message en entrée. Après tout, au niveau le plus bas, l'algorithme n'effectue rien d'autre qu'une suite d'opérations logiques purement booléennes sur les mots du paquet transmis. La conclusion de cette étude est la suivante : en fournissant des contenus

spécifiques au paquet ("entrée"), le système distant peut alors être contraint de procéder à une série d'opérations arithmétiques, puis d'évaluer son exactitude – donner son accord avec la somme de contrôle déclarée dans l'en-tête TCP ou IP.

Bien que l'opération exécutée par le système distant pendant le processus de la somme de contrôle soit exactement le même à chaque itération, il dispose d'une fonctionnalité suffisante pour servir de porte logique universelle, un mécanisme que nous avons abordé au Chapitre 2. En intercalant dans l'entrée effectivement testée des mots de "contrôle" soigneusement choisis qui inversent ou modifient la somme de contrôle partielle calculée jusque-là, il est possible de mener à bien n'importe quelle opération booléenne.

Cela signifie que la logique SAT peut être facilement recréée en utilisant une séquence de contrôle spécifique et des bits en "entrée"dans un paquet une fois que les données sont exposées à l'algorithme de la somme de contrôle ; les variables de l'équation (dans un sens ou dans l'autre) sont intercalées avec des mots définis qui sont utilisés pour transformer la valeur actuelle de la somme de contrôle afin que la sortie de la prochaine opération imite un opérateur booléen spécifique. Le résultat final – la valeur à laquelle un paquet se résume – montre la sortie finale : la valeur logique d'une formule à évaluer.

Ainsi, le test de valuation est accidentellement effectué par le destinataire distant lorsqu'il tente de valider la somme de contrôle à son arrivée. Si la somme de contrôle est égale à 1 (ou à quelque autre valeur qui correspond dans notre système de calcul SAT à une déclaration SAT dont l'évaluation est vraie), le test de valuation des valeurs de variable choisies pour ce paquet réussit (le trafic est transmis aux couches supérieures et une action s'ensuit). Si la somme de contrôle échoue, la formule n'a pas été remplie, et le paquet est abandonné en silence. En d'autres termes, si nos bits en entrée émettent une hypothèse, le destinataire vérifie qu'elle est vraie ou fausse et agit différemment en fonction du résultat.

En outre, une partie qui veut résoudre un problème SAT rapidement peut préparer un ensemble de toutes les combinaisons de valeurs possibles que les variables (entrées) peuvent avoir dans une formule donnée, intercaler ces valeurs avec les informations qui font que les entrées se combinent avec les autres données de la meilleure façon, placer ces informations dans des paquets TCP et les envoyer (presque en parallèle) à un grand nombre d'hôtes dans le monde entier. Au lieu d'être calculée, la somme de contrôle pour un paquet serait définie manuellement à une valeur que nous savons correspondre à ce que "l'hypothèse" produirait si elle s'avérait. Seuls les hôtes qui reçoivent des paquets contenant des valeurs de variables avec lesquelles l'évaluation de la formule donne la valeur souhaitée répondront au trafic ; les autres systèmes considéreront tout simplement ce trafic comme corrompu en raison de l'inadéquation de la somme de contrôle. L'expéditeur peut ainsi déterminer la solution correcte, sans effectuer de calculs massifs. Il lui suffit de vérifier l'ensemble des valeurs utilisées dans les paquets envoyés aux hôtes qui ont répondu à une requête.

L'étude va plus loin et signale une tentative réussie de résoudre un problème NP en utilisant des hôtes réels dans le monde entier, fournissant donc non seulement un fondement théorique mais aussi la confirmation réelle du bien-fondé de cette approche.

Cette technique a un impact très léger, mais également important : elle prouve qu'il est effectivement possible de "délocaliser" les calculs vers des parties distantes sur le réseau sans qu'elles ne le sachent et ne le veuillent, y compris certaines opérations nécessaires pour résoudre de vrais problèmes de calcul. Et cela peut être réalisé sans réellement attaquer ces systèmes, en prendre le contrôle, installer de logiciels malveillants ni interférer en quoi que ce soit avec les tâches légitimes. Une personne peut donc effectivement diviser une tâche de calcul entre un grand nombre de systèmes. Ce processus ne consomme qu'une partie infime et négligeable de la puissance de calcul de chaque système mais peut néanmoins correspondre à l'équivalent d'un superordinateur quand des millions de systèmes collaborent sur un même problème.

La domination du monde à portée de main ? Pas si vite.

Considérations pratiques

… Ou peut-être, mais pas seulement. L'approche suggérée dans cette recherche est révolutionnaire et intéressante mais ne représente pas nécessairement une manière particulièrement pratique de construire un superordinateur en volant aux riches. Le montant de bande passante nécessaire pour maintenir un taux raisonnable de calcul ainsi que le nombre de calculs nécessaires pour préparer des problèmes simples que les autres systèmes doivent résoudre sont assez élevés. De ce fait, ce système n'est pas assez efficace pour externaliser la résolution de problèmes mathématiques complexes sur les systèmes de victimes non consentantes.

Dans le schéma décrit plus haut, la puissance de calcul exponentielle nécessaire est remplacée par la bande passante exponentielle nécessaire. Ce n'est pas forcément un échange intéressant, surtout si l'on considère que des tests relativement simples peuvent ne pas se dérouler compte tenu de la taille limite des paquets sur la plupart des réseaux (tous ces tests pourraient facilement être résolus en autant de temps qu'il n'en faut pour transmettre ces données sur Ethernet). Cette technique apporte la preuve que l'attaque est possible et fournit un moyen véritablement universel de la mener à bien, mais des scénarios d'attaque plus spécifiques pourraient donner des résultats beaucoup plus utiles.

D'autres moyens de parvenir à une puissance de calcul impressionnante à moindre coût en volant des quantités négligeables de la puissance de calcul de chaque machine sont peut-être plus intéressants. Par exemple, certains types de logiciels clients (comme les navigateurs Web) peuvent facilement être utilisés pour exécuter des algorithmes assez

complexes d'une façon relativement simple. Sur son site md5crk.com, Jean-Luc Cooke encourageait par exemple les webmasters à ajouter à leurs pages Web un petit applet Java qui utilisait un schéma de calcul basé sur la "loterie chinoise" et détaillé dans la RFC 3607[2]. Une fois cet applet ajouté à un site, chaque visiteur pouvait exécuter l'applet sur son système. L'applet empruntait une quantité négligeable des cycles du processeur pour contribuer à un projet visant à trouver des collisions dans les fonctions des résumés MD5 (les collisions désignent deux messages différents qui produisent le même résumé. Ces cas rares et anecdotiques mais très certainement possibles* peuvent nous permettre de mieux comprendre les faiblesses des fonctions de résumé et pourraient empiriquement prouver, voire démontrer, que MD5 ne représente plus un écueil pour les ordinateurs d'aujourd'hui).

Les applets Java sont de petits programmes indépendants de la machine qui sont exécutés par défaut par les navigateurs Web dans des environnements "sandbox" spéciaux et restreints. Ils n'ont pas accès au contenu local du disque et ne peuvent (en théorie seulement) faire aucun mal, même s'ils peuvent utiliser la connectivité du réseau de façon limitée pour effectuer des calculs et ajouter certains éléments visuels à une page Web. Ils sont le plus couramment utilisés pour améliorer les sites Web en leur apportant des fonctionnalités supplémentaires comme des jeux interactifs, des effets visuels, et ainsi de suite. Mais Jean-Luc utilisait ces applets pour quelque chose d'autre : pour trouver des collisions possibles en utilisant simultanément la puissance de calcul conjointe de centaines ou de milliers de systèmes à travers le monde.

Le principe de cette opération était simple : l'applet était exécuté sur les systèmes clients du monde entier chaque fois qu'un site coopérant à l'opération était visité. Ensuite, une fois lancé, l'applet essayait de calculer le résumé MD5 de différents messages choisis au hasard. Cela continuait jusqu'à ce que soit trouvé un résumé qui corresponde à un certain motif de masquage arbitrairement choisi et défini. Ce motif pouvait être "tout résumé dont les quatre derniers octets valent zéro" ou quelque chose de semblable. Le motif était choisi de manière que la découverte du résumé par tâtonnements ne prenne pas trop de temps (de sorte que la personne n'ait pas à quitter la page Web et à arrêter le code avant que le résumé ne soit trouvé) mais aussi de manière qu'une petite fraction seulement de l'ensemble des résumés possibles corresponde.

---

* Alors que la version originale de ce livre était en cours de préparation pour l'impression, une équipe de chercheurs chinois de l'université de Shandong (Xiaoyun Wang, Dengguo Feng, Xuejia Lai et Hongbo Yu) ont découvert une technique pour trouver des collisions MD4, MD5, Haval-128 et RIPEMD-128 et ont fourni des échantillons. Cette découverte majeure pour la cryptographie moderne confirme que ces fonctions sont insuffisantes pour certaines applications liées à la sécurité. Bien que le projet md5crk.com se soit interrompu, sa contribution à l'exploration du calcul parasitaire reste valable.

Une fois qu'un message était trouvé, le programme le signalait et l'envoyait à l'auteur, qui pouvait alors l'examiner. L'applet avait déjà examiné et rejeté un certain nombre de possibilités de collision et transmettait uniquement celles qui correspondaient à une condition prédéfinie (celles qui étaient en partie identiques). Comme les données collectées de cette façon risquent beaucoup moins de varier, le risque de collision dans un morceau de *n* entrées est considérablement plus élevé que pour des données purement aléatoires. Par analogie, la probabilité de rencontrer deux pommes ayant le même aspect dans un lot est plus élevée si l'on commande uniquement des pommes qui ont à peu près le même poids et la même couleur que si l'on achète des fruits en vrac.

Bien que discutable d'un point de vue éthique, cette approche ingénieuse déployée pour la première fois par md5crk.com a vraiment fonctionné et a démontré que le calcul parasitaire peut être à la fois très efficace et furtif. Il est possible de voler des cycles processeur dans un but "légitime" et peut-être même de les utiliser plus souvent que nous le voudrions. Et cette possibilité est là pour perdurer.

Si on est sceptique, on peut se demander si le calcul parasitaire peut utiliser un peu de la puissance des processeurs pour autre chose que le déchiffrement des systèmes de cryptage, une tâche qui n'intéresse que peu de personnes.

Les débuts du stockage parasitaire

Lorsque vous criez, les ondes sonores se déplacent dans l'air, perdent peu à peu de l'énergie et se dispersent dans toutes les directions. Toutefois, si elles rencontrent un obstacle solide, elles vont probablement rebondir et, si l'angle est le bon, revenir vers vous. Vous pouvez constater ce résultat sonore lorsque vous entendez l'écho de votre voix une fraction de seconde après avoir crié.

Mais que se passe-t-il quand un fanatique des théories de l'information lit à haute voix son code depuis le sommet d'une montagne, en dirigeant ses paroles vers une vallée rocheuse ? J'attendais que vous le demandiez. Dans ce cas, il ne peut que faire le constat suivant : s'il lit rapidement puis oublie immédiatement tout ce qu'il a récité (parce que d'autres questions le préoccupent), il peut encore éventuellement récupérer les informations lorsqu'elles se répercutent depuis le bas de la vallée et font écho. Voilà un mécanisme de stockage de données commode.

Cela vous semble ridicule ? Peut-être que nous sommes tout simplement trop jeunes. Les premiers types de modules de mémoire informatique utilisaient une technique acoustique similaire pour permettre au processeur de stocker des informations "hors ligne" et les récupérer plus tard. Plutôt qu'utiliser l'air (à travers lequel les ondes se propagent un peu trop rapidement pour fournir des capacités de stockage suffisantes sans devoir créer des unités de mémoire de très grande taille), un tambour rempli de mercure était utilisé (un environnement dans lequel les ondes acoustiques se propagent beaucoup

plus lentement). Toutefois, le principe restait identique et donna même un sens intéressant à la notion de fuite de mémoire. Ce dispositif de mémoire à mercure a par exemple été utilisé dans le célèbre UNIVAC I*.

Naturellement, ce type de mémoire lente, encombrante, dangereuse et peu pratique a été abandonné au profit d'autres solutions, dès que la technologie a évolué. Toutefois, l'invention elle-même avait un certain charme et ne sera pas oubliée si facilement. Lors de la conférence DefCON de Las Vegas, en 2002, Saqib A. Khan fit une courte présentation sur ce sujet en indiquant comment utiliser les propriétés d'un réseau de grande taille comme Internet pour construire le même type de stockage momentané. Pour tous les hackers et les geeks qui regardaient ce court diaporama, la description de la mémoire acoustique n'a pas semblé ridiculement primitive mais les a fascinés. La mémoire acoustique fit son retour avec élégance.

Comme les temps d'aller-retour des paquets (le temps nécessaire pour qu'un message arrive à un système distant et pour qu'une réponse revienne) ne sont pas nuls, une certaine quantité de données peut toujours être conservée "sur le fil" en envoyant et en recevant de façon répétée des portions de ces données et en attendant qu'elles produisent un écho en retour. Saqib utilisait des paquets "echo request" (ping) du protocole ICMP (*Internet Control Message Protocol*) pour obtenir cet effet. La plupart des systèmes sur Internet répondent à de tels paquets avec "echo reply", en citant la charge utile originale qu'ils ont reçue.

Cela semblait une astuce intelligente. Toutefois, la mise en application de cette technique est également loin d'être pratique car elle exige la retransmission fréquente de portions de données. Comme le message ICMP "echo reply" est renvoyé presque immédiatement après la réception du message "echo request", seule une petite quantité de données peut être transmise à chaque aller-retour. En conséquence, la quantité de données pouvant être stockées de cette façon ne saurait dépasser ce que l'utilisateur peut transmettre pendant quelques secondes (le plus souvent, pendant moins d'un dixième de seconde).

Ah ! Mais le stockage parasitaire pourrait être amélioré.

---

* On peut noter qu'une mémoire analogique de faible capacité a également été utilisée au début des implémentations des récepteurs de télévision SECAM (séquentiel couleur avec mémoire). Contrairement au NTSC ou au PAL, le SECAM utilise une résolution colorimétrique moindre ; les composantes de chrominance rouges et bleues sont transmises en alternance, jamais simultanément. L'autre composante doit être prise à partir de la ligne précédente pour déterminer l'aspect d'un pixel en particulier. Pour ce faire, un dispositif de mémoire doit être implémenté.

Rendre possible le stockage parasitaire

En 2003, Wojciech Purczynski et moi avons coécrit un document appelé "Juggling with Packets: Parasitic Data Storage" (Jongler avec les paquets : le stockage parasitaire des données). Nous avons poussé le concept de stockage parasitaire un peu plus loin et examiné un certain nombre de méthodes qui pourraient être utilisées pour étendre de manière spectaculaire la capacité de stockage d'Internet, tout en préservant la bande passante nécessaire pour maintenir l'information. Nos recherches ont porté sur plusieurs autres manières de stocker des données sur des systèmes distants que nous avons classées en fonction des propriétés du support de stockage (sa visibilité, sa volatilité et sa fiabilité). Nous avons également inclus un examen détaillé de capacités de stockage hypothétiques de chacune de ces techniques.

Ce document très court, amusant et rafraîchissant (du moins, je l'espère) est reproduit ici.

```
============================================================
Jongler avec les paquets : le stockage flottant de données
============================================================

"Votre donjon est construit sur une pente. Les monstres ne peuvent pas
jouer aux billes !"

Wojciech Purczynski <cliph@isec.pl>
Michal Zalewski <lcamtuf@coredump.cx>

1) Jongler avec des oranges !
-----------------------------

La plupart d'entre nous, y compris les auteurs de ce document, ont
tenté de jongler avec trois pommes ou plus, des oranges ou d'autres
objets fragiles. L'effet est en général assez pathétique, mais le plus
habile des apprentis jongleurs apprend tôt ou tard à le faire sans
entraîner de dommages collatéraux excessifs.

S'il est observateur, un apprenti jongleur peut remarquer que, tant
qu'il continue à suivre une procédure simple, au moins un des objets
est toujours en l'air et qu'il doit tenir deux objets au maximum en
même temps. Pourtant, chaque pomme passe de temps en temps dans ses
mains et il peut la récupérer quand il le souhaite.

Après s'être un peu amusé à jongler, notre apprenti jongleur décide que
l'ensemble du processus est extrêmement ennuyeux et retourne à son
ordinateur. Tout en vérifiant son courrier électronique, il remarque
qu'un service réseau typique n'a qu'une occupation : accepter et
traiter les données provenant d'un système distant et prendre toutes
mesures qu'il juge appropriées en fonction de son interprétation des
données. Beaucoup de ces services font de leur mieux pour être
robustes, tolérants aux pannes et fournir des indications utiles sur la
transaction.
```

Dans certains cas, le simple fait que le service tente de traiter les données et de répondre peut être utilisé d'une façon que les auteurs du protocole n'ont jamais imaginée. L'étude intitulée "Parasitic Computing" (calcul parasitaire) de l'université de Notre Dame et publiée dans des lettres au journal *Nature* en est un spectaculaire exemple.

Néanmoins, notre héros conclut que ces tentatives sont assez peu pratiques dans le monde réel. Le coût de la préparation et de la livraison des problèmes simples à résoudre dépasse de loin les gains éventuels puisque l'expéditeur doit effectuer des opérations aussi complexes pour émettre la requête que s'il effectuait les calculs lui-même. Pour lui, "la puissance de calcul d'un tel dispositif est minable !".

Un vrai jongleur se concentrerait alors sur une forme d'externalisation du traitement de données qui soit beaucoup plus proche de son domaine d'expertise. Pourquoi ne pas implémenter un stockage des données fondé sur le mouvement des fruits ? Que se passe-t-il si j'écris une seule lettre sur chaque orange et que je commence ensuite à jongler ? Je peux alors enregistrer plus d'octets d'orange que ma capacité physique ne le permet (le nombre d'oranges que je peux tenir dans mes mains) ! Génial… Mais cela fonctionnerait-il sans les oranges ?

2) La même chose, sans les oranges

Le présent document se fonde sur l'observation suivante : pour l'ensemble des communications réseau, il y a toujours un délai différent de zéro (et souvent considérable) entre l'envoi des informations et la réception d'une réponse du fait des contraintes physiques du médium utilisé pour le transfert et le temps nécessaire au traitement des données sur tous les équipements informatiques.

De la même façon qu'une orange sur laquelle est écrit un message, un paquet utilisé pour stocker des données voyage pendant un certain temps avant de revenir à sa source ; durée pendant laquelle nous pouvons oublier le message sans perdre de données. En tant que tel, Internet a une capacité de stockage momentanée de données non nulle. Il est donc possible de transmettre une information et qu'elle soit effectivement stockée jusqu'à ce qu'elle soit renvoyée en écho. En établissant un mécanisme pour la transmission et la réception cyclique de morceaux de données vers et depuis un certain nombre d'hôtes distants, il est possible de maintenir un nombre arbitraire de données constamment "sur le fil" et donc d'établir ainsi un médium volatile de grande capacité.

Ce support peut être utilisé pour des opérations coûteuses en mémoire, soit comme stockage classique soit pour certains types de données sensibles pour lesquelles on ne souhaite pas laisser de traces physiques sur un disque dur ou tout autre support non volatile. Puisqu'on ne considère pas que renvoyer autant d'informations pertinentes à l'expéditeur que l'expéditeur en envoie au service soit une erreur de programmation et puisque de nombreux services et piles

```
maintiennent un haut niveau de verbosité, notre expérience de la
jonglerie nous indique qu'il est non seulement possible mais aussi
réalisable de mettre en place ce type de stockage, même sur un réseau à
bas débit. Contrairement aux méthodes traditionnelles de stockage
parasitaire de données (comme l'utilisation abusive du P2P, les
serveurs FTP ouverts, les fichiers binaires Usenet, etc.), cette
méthode peut ou peut ne pas laisser de trace des données (selon la
façon dont nous l'implémentons) et ne crée aucune charge particulière
sur aucun système. En conséquence, contrairement aux méthodes
traditionnelles, cette technique risque moins d'être détectée et d'être
considérée comme un abus. Ainsi, la possibilité que les données soient
interceptées et délibérément supprimées est beaucoup moins
problématique.

3) Stockage des données de classe A : la mémoire tampon
-------------------------------------------------------

Le stockage des données de classe A utilise les capacités inhérentes
aux délais des communications durant la transmission et le traitement
des données en temps réel lorsqu'elles traversent des réseaux entre
deux systèmes d'extrémité. Les informations stockées demeurent dans la
mémoire cache d'un ordinateur distant et ne sont pas susceptibles
d'être transférées sur un disque physique.

Tous les exemples de mémoire de classe A reposent sur l'envoi d'un
message connu pour entraîner un écho partiel ou total à la demande
initiale. Cet écho peut être obtenu par :

   - Les réponses SYN+ACK, RST+ACK aux paquets SYN, et par d'autres
     rebonds.

   - Les réponses "echo reply" ICMP.

   - Les réponses DNS lookup et les données mises en cache. Il est
     possible de stocker des informations dans une requête lookup et de
     les renvoyer avec une réponse NXDomain ou de stocker les données
     dans une mémoire cache NS.

   - Le transfert des messages de discussion entre plusieurs serveurs.
     Relayer les messages textuels à travers les serveurs IRC et autres
     entraîne une latence considérable.

   - HTTP, FTP, proxy Web, erreur SMTP ou les réponses d'état.

Les propriétés les plus importantes de la classe A de stockage sont les
suivantes :

   - Une faible latence (de quelques millisecondes à plusieurs
     minutes), ce qui la rend plus utile pour les applications proches
     de la mémoire vive.
   - Une capacité de stockage par système faible (généralement quelques
     kilo-octets), ce qui la rend inadaptée au stockage de fichiers de
     grande taille.
```

- Une seule chance d'être reçue ou peu de chances d'être retransmise, ce qui la rend moins fiable en cas de panne du réseau.

- Une réduction des risques d'enregistrement permanent. Les données ne sont pas susceptibles d'être stockées sur un support non volatile ou déplacées sur une mémoire à accès lent, ce qui accroît leur anonymat et rend plus facile de nier qu'elles nous appartiennent.

Lors de l'utilisation de protocoles de plus haut niveau, des caractéristiques supplémentaires apparaissent qui pourraient résoudre certains problèmes (temps court pour récupérer les données, faible capacité) communs aux différents types de stockages de classe A. Il est par exemple possible d'établir une connexion à un service comme SMTP, FTP, HTTP ou à tout autre service fondé sur le texte et d'envoyer une commande connue pour provoquer l'envoi en écho d'un message d'erreur ou d'acquittement contenant une partie des données initiales. Toutefois, on n'envoie pas un message entièrement formaté ; certains caractères ne doivent pas être envoyés. Dans la plupart des cas, les caractères de fin de ligne sont nécessaires pour mener à bien la commande. Dans cette phase, nos données sont déjà stockées sur le service distant qui attend une commande complète ou que le temps de connexion expire. Pour ne pas dépasser ce délai, que ce soit sur TCP ou au niveau de l'application, des paquets no-op doivent être envoyés périodiquement. Un caractère \0 interprété comme une chaîne vide ne produit aucun effet sur de nombreux services, mais il suffit pour réinitialiser les compteurs TCP et le délai d'attente du service. Microsoft Exchange est un bon exemple d'une application vulnérable à cette attaque.

L'attaquant peut maintenir la connexion pendant une durée arbitraire avec une portion de données déjà stockées sur l'autre système d'extrémité. Pour récupérer les informations, la commande doit être complétée avec les caractères \r\n manquants, puis la réponse est envoyée au client.

La commande SMTP VRFY est un bon exemple :

```
220 inet-imc-01.redmond.corp.microsoft.com Microsoft.com ESMTP
Server
Thu, 2 Oct 2003 15:13:22 -0700
VRFY AAAA ...
252 2.1.5 Cannot VRFY user, but will take message for
<AAAA...@microsoft.com>
```

Il est possible de stocker un peu plus de 300 octets, y compris les caractères non imprimables, de cette façon et de les rendre disponibles presque instantanément. Davantage de données peuvent être stockées si la méthode HTTP TRACE est utilisée avec les données transmises dans des en-têtes HTTP arbitraires, selon le logiciel du serveur. En maintenant les connexions, on peut obtenir une latence arbitrairement élevée et donc créer une plus grande capacité de stockage.

Ce type de stockage est naturellement plus adapté aux applications dans lesquelles la confidentialité est primordiale ou lorsque la latence la plus faible est inférieure à la capacité de stockage du médium (stockage immédiat sur la RAM des informations qui ne doivent laisser aucune trace visible). Ce stockage n'est pas adapté aux données essentielles qui devront être préservées à tout prix, en raison du risque que les données soient perdues en cas de panne du réseau.

4) Stockage des données de classe B : les files d'attente du disque
---

Le stockage des données de classe B utilise les files d'attente de données "inactives"qui stockent des informations pendant une longue période de temps (souvent sur le disque). Par exemple, les systèmes MTA peuvent mettre en file d'attente les courriers électroniques pendant sept jours (ou plus, selon la configuration). Cette fonctionnalité peut nous donner un long délai entre l'envoi de données à stocker sur l'ordinateur hôte et leur réception. Comme un serveur SMTP typique empêche que le courrier électronique du client soit relayé au serveur lui-même, on peut utiliser les rebonds de l'e-mail pour que les données reviennent après une longue période de temps.

Prenons par exemple ce scénario potentiel d'attaque :

1. L'utilisateur construit une liste de serveurs SMTP (peut-être des serveurs qui semblent être hors de portée de ses adversaires).

2. L'utilisateur bloque (avec bloc/drop, et non pas reject) toutes les connexions entrantes sur le port 25.

3. Pour chaque serveur, l'attaquant doit confirmer le délai d'expiration de la livraison et l'adresse IP à partir de laquelle le serveur se connecte lorsqu'il essaie de renvoyer un rebond. Cela se fait par l'envoi d'une sonde à une adresse locale au serveur (ou en demandant une notification DSN pour une adresse valide) et en vérifiant pendant combien de temps le serveur essaie de se connecter avant d'abandonner. Le serveur n'a pas besoin d'être un relais ouvert.

4. Après avoir confirmé les objectifs, l'attaquant commence à envoyer des données à un rythme choisi pour que le processus soit réparti uniformément sur une période d'une semaine. Les données doivent être divisées de manière qu'il y ait un morceau pour chaque serveur. Chaque morceau est envoyé à un serveur distinct pour générer immédiatement un rebond renvoyé à l'expéditeur.
5. Le processus de maintien de données revient à accepter une connexion entrante et à recevoir le retour une semaine au plus tard à compter de la date d'envoi initiale, juste avant que l'entrée ne soit retirée de la file d'attente. Cela se fait en permettant à ce serveur en particulier de passer à travers le pare-feu. Dès que le morceau est reçu, il est relayé en retour.

6. Pour accéder à toute portion de données, l'attaquant cherche quel MTA détient ce bloc puis autorise cette adresse IP à se connecter et à envoyer le rebond. Trois scénarios sont alors possibles :

   - Si le MTA distant prend en charge la commande ETRN, la livraison peut être initiée immédiatement.

   - Si le MTA distant exécutait depuis trois minutes une tentative de connexion à un système local (il continue d'essayer grâce au fait que ses paquets SYN sont abandonnés et non rejetés avec RST+ACK), la connexion peut être établie en l'espace de quelques secondes.

   - Sinon il est nécessaire d'attendre de cinq minutes à une heure, en fonction des paramètres de la file d'attente.

Ce motif peut être amélioré en utilisant des noms DNS au lieu des adresses IP pour les utilisateurs ayant une adresse IP dynamique ou pour fournir une protection supplémentaire (ou lorsqu'il est nécessaire de couper la chaîne immédiatement).

Les propriétés importantes de stockage de classe B sont les suivantes :

- Capacité élevée par système (plusieurs mégaoctets), ce qui en fait la solution idéale pour stocker des gros fichiers, par exemple.

- Une latence d'accès plus élevée (quelques minutes à quelques heures), plus proche d'un périphérique à bande que de la RAM (à l'exception des hôtes SMTP, qui acceptent que la commande ETRN tente de nouveau d'initier un transfert immédiat).

- Une très longue durée de vie, ce qui augmente la capacité et la fiabilité de chaque utilisateur.

- Beaucoup de tentatives de livraison, ce qui facilite la récupération des données même après un problème réseau ou matériel temporaire.

- Un risque certain de laisser une trace sur les périphériques de stockage, ce qui en fait une solution moins utile pour le stockage dont on nie totalement la paternité (même s'il serait nécessaire d'examiner un certain nombre de systèmes dans différents pays, ce qui ne doit pas être réalisable).

Le stockage de classe B est adapté pour stocker des archives de fichiers ordinaires, les mémoires tampon de grande taille en mode append-only, les ressources cryptées (avec une bonne sélection des hôtes, il est quasiment possible de nier leur paternité), etc.

```
5) Stockage de classe A discret
--------------------------------
Dans certaines situations, il peut être nécessaire de concevoir une
solution pour le stockage discret de données qui ne réside pas sur la
machine elle-même et qui permette de nier la présence de cette
information à un endroit ou à un autre.

L'exigence de base est que les données soient :

   - Non renvoyées jusqu'à ce qu'une séquence clé spéciale soit
     envoyée.

   - Supprimées de façon permanente sans laisser aucune trace sur
     n'importe quel support de stockage non volatile en l'absence de
     requête keep-alive.

Il est possible d'utiliser le stockage de classe A pour implémenter
cette fonctionnalité en utilisant la méthode de commande pour maintenir
les données dont nous avons parlé plus tôt. Le bon numéro de séquence
TCP est nécessaire pour libérer les données et, tant que cette séquence
n'est pas fournie, les données ne sont pas renvoyées ou divulguées à
aucune partie. Si le nœud client est hors ligne, les données sont
supprimées et probablement écrasées par d'autres.

Le numéro de séquence est donc la clef à l'information stockée et, si
la durée de vie restant aux données est assez courte quand la commande
d'arrêt du keep-alive \0s arrive, il s'agit souvent d'une protection
adéquate.

6) La capacité accessible à l'utilisateur
------------------------------------------

Dans cette section, nous tentons d'estimer la capacité de stockage
disponible pour un seul utilisateur.

Afin de maintenir un niveau constant de données "externalisées" sur le
réseau, nous devons être en mesure de les recevoir et de les renvoyer
régulièrement.

Le temps pendant lequel les données peuvent être stockées à distance
est limité par la durée de vie maximale Tmax d'un seul paquet (mise en
file d'attente et délais de traitement du paquet compris). La quantité
maximale de données pouvant être envoyées est limitée par la bande
passante maximale du réseau qui est disponible (L). Ainsi, la capacité
maximale peut être définie comme suit :

   Cmax [octets] = L [octets / seconde] * Tmax [secondes] / Ptaille *
   Dtaille

avec :

Dtaille - La taille d'un paquet nécessaire pour stocker une première
partie des données sur un hôte distant.
```

Ptaille - La taille d'un paquet nécessaire pour maintenir les informations stockées sur un hôte distant.

Ptaille et Dtaille sont égaux et peuvent donc être omis lorsque la totalité du bloc de données effectue des rebonds ; ils ne diffèrent que pour les cas où la "commande de maintien" des données est utilisée. Le plus petit paquet TCP/IP pour accomplir cela compte 41 octets. La quantité maximale de données pouvant être maintenue en utilisant les en-têtes HTTP est d'environ 4 096 octets.

Ce qui nous donne le tableau suivant :

Bande passante	Classe A	Classe B
28,8 Kbits/s	105 Mo	2 Go
256 Kbits/s	936 Mo	18 Go
2 Mbit/s	7,3 Go	147 Go
100 Mbits/s	365 Go	7 To

7) Internet dans son ensemble

Dans cette section, nous essayons d'estimer la capacité théorique momentanée d'Internet dans son ensemble.

Classe A

Pour estimer la capacité théorique du stockage de classe A sur Internet, nous supposons que :

- Les messages ICMP offrent le meilleur équilibre entre capacité de stockage et préservation des ressources sur un système distant.

- Sur un système d'exploitation, la file d'attente des paquets d'entrée peut en moyenne contenir au moins 64 paquets.

- La valeur par défaut du PMTU est d'environ 1 500 (le MTU le plus courant).

Pour estimer le nombre d'hôtes sur Internet, nous utilisons une enquête de l'ISC (Internet Systems Consortium) réalisée en 2003, qui compte 171 638 297 systèmes ayant des entrées DNS inversées (bien que toutes les adresses IP ayant des DNS inversées ne soient pas obligatoirement opérationnelles). Pour en tenir compte, nous avons utilisé le taux de réponse à l'écho ICMP calculé par la dernière enquête à avoir réalisé ce test (en 1999). Les données indiquent qu'environ 20 % des systèmes visibles étaient opérationnels, ce qui signifie que le nombre de systèmes prêts à répondre aux requêtes ICMP est d'environ 34 millions.

```
En multipliant le nombre de systèmes qui répondent aux requêtes écho
ICMP par la taille moyenne du cache des paquets et la taille maximale
des paquets (moins les en-têtes), nous estimons que la capacité
théorique totale du stockage ICMP momentané de classe A s'élève à
environ 3 To.

Classe B

Pour estimer la capacité de stockage théorique de la classe B, nous
utilisons l'exemple du logiciel MTA. Il n'y a pas de limite à la
quantité de données que nous pouvons fournir à un seul hôte. Bien qu'il
soit raisonnable de penser que seuls les messages d'une taille
inférieure à 1 Mo environ ne causeront aucune charge sur le système qui
puisse être constatée ni aucun autre effet indésirable, nous supposons
que la taille moyenne de la file d'attente est de 500 Mo.

Nos propres recherches indiquent que le port 25 est ouvert sur près de
15 % des systèmes qui répondent aux requêtes Ping. Nous avons donc
évalué la population des serveurs SMTP à 3 % (15 % de 20 %) du montant
total des hôtes, soit un peu plus de 5 millions d'hôtes.

Cela donne une capacité totale de stockage de 2 500 To.
```

Applications, considérations sociales, et défense

Et maintenant ? À quoi servent le calcul parasitaire et les systèmes de stockage si les avantages qu'ils procurent sont encore loin d'être suffisants pour représenter une alternative séduisante à l'augmentation de la puissance matérielle ?

Malgré les progrès de l'exploitation pratique du calcul parasitaire, les applications qui visent à étendre la puissance de calcul ou l'espace de stockage d'un système traditionnel peuvent sembler futiles si l'on considère l'abondance de la mémoire à bon marché et des processeurs dont la fréquence dépasse le gigahertz.

Le vrai potentiel de cette technologie peut toutefois se trouver dans un tout autre ensemble d'applications : le *calcul volatil*. Construire des ordinateurs distribués utilisables qui peuvent se disperser à volonté, sans laisser de traces physiques et sans stocker de données significatives nulle part, peut être un puissant outil de protection de la vie privée, même si cela représente également un défi pour les enquêtes judiciaires et l'application des lois. Comme la mémoire volatile stocke et conserve des données sans qu'il soit obligatoire de retransmettre fréquemment des données mais se vide en peu de temps si un seul nœud est hors ligne, l'auteur d'une infraction (ou le membre d'une minorité opprimée, d'ailleurs) peut facilement nier que ces données lui appartiennent. Par conséquent, un grand nombre de procédures de recherche de preuves doivent être modifiées assez radicalement.

En outre, imaginez des systèmes volatiles qui pourraient, une fois amorcés et initialisés, se maintenir pendant de longues périodes de temps et vivre dans Internet sans consistance

physique. Deux conceptions sont possibles pour les systèmes informatiques volatils et distribués, et aucune n'est totalement absurde :

- Les systèmes peuvent être conçus de manière à remplir une tâche complexe en recherchant une solution en parallèle (ce que réalise déjà en grande partie le système informatique SAT, dont nous avons parlé plus tôt). Ces systèmes présentent deux inconvénients : le résultat du calcul doit être récupéré et la prochaine itération du traitement doit être lancée manuellement par l'ajout occasionnel de "nouveaux germes" à l'ensemble du système depuis un endroit quelconque. Les solutions qui reposent sur les propriétés de bas niveau des protocoles comme TCP entrent probablement dans cette catégorie.
- Les systèmes peuvent être conçus de façon à exécuter eux-mêmes des suites d'itérations de calcul distribué. Tous les types d'abus des fonctionnalités de plus haut niveau (comme les algorithmes de rendu intégrés aux documents) et certains services de réseau peuvent être utilisés pour faciliter ce type d'activité.

Dans chaque cas, les conséquences peuvent être assez importantes. Par exemple, comment prendre le contrôle d'une machine qui se répare elle-même et qui n'utilise aucun système en particulier mais emprunte de minuscules bribes de mémoire et de puissance de traitement aux autres pendant une fraction de seconde sans utiliser les failles de ces systèmes ni produire de trafic clairement identifiable qui puisse être filtré ? N'est-il pas aussi un peu inquiétant de constater que nous ne serions pas en mesure de discerner immédiatement les objectifs d'un tel ordinateur distribué ? Je m'incline respectueusement devant les maîtres de la mauvaise science-fiction car je pense que la domination des ordinateurs est imminente. Je souhaite la bienvenue à nos nouveaux maîtres, les machines.

Matière à réflexion

Il est en général extrêmement difficile de se défendre contre le calcul parasitaire. La capacité de stocker des données ou de forcer l'autre partie à procéder à certains calculs simples est souvent liée aux fonctionnalités fondamentales des protocoles réseau. Il est impossible d'imaginer supprimer cette caractéristique sans anéantir Internet tel que nous le connaissons ni introduire une foule de nouveaux problèmes plus graves que ceux auxquels on remédierait.

Il est également assez difficile d'empêcher qu'un seul système devienne un nœud de calcul parasitaire. Comme le nombre de ressources volées à un système représente souvent une part négligeable de l'inactivité du temps et de la mémoire du processeur, cette technique peut par conséquent facilement passer inaperçue.

Il est probable que le calcul parasitaire n'ait pas encore révélé tout son potentiel et que cette menace, qui concerne non pas les systèmes en particulier mais le réseau dans son ensemble, soit amenée à perdurer.

17

La topologie du réseau

Ou comment connaître le monde qui nous entoure
peut nous aider à suivre la trace des amis et des ennemis.

Quelle est la forme d'Internet ? Aucune commission ne le supervise ni ne décide où, comment et pourquoi il doit s'étendre ni encore comment les nouveaux systèmes et les systèmes existants doivent être organisés ou gérés. Le réseau Internet se développe dans toutes les directions en fonction de la demande, pour des raisons économiques, politiques, techniques ou au hasard.

Cependant, Internet n'est pas une entité informe : il existe une hiérarchie des systèmes autonomes planifiée et régie au niveau local, avec des routeurs de base entourés de nœuds de moindre importance, des liaisons configurées par des mécanismes automatiques ou soigneusement conçues par l'homme. Internet est une structure maillée spectaculaire, une toile d'araignée à la fois complexe et fragile qui couvre l'ensemble du monde industrialisé et en voie de développement. Il semble difficile de prendre un instantané de cette topologie sans cesse changeante, mais il est aussi tentant d'essayer de le faire, en particulier lorsque nous nous rendons compte à quel point nous pourrions tirer parti de cette information.

Dans ce chapitre, je vais d'abord examiner deux tentatives notables visant à dresser la carte de la topologie du réseau. Je vais ensuite une fois de plus considérer les utilisations possibles de cette information d'un point de vue moral.

Capturer l'instant

La tentative la plus complète d'établir la carte d'Internet a été effectuée par la CAIDA (Cooperative Association for Internet Data Analysis), une organisation financée entre autres par des agences de recherche gouvernementales américaines (NSF, DHS, DARPA) et des entreprises (Cisco, Sun). Cette organisation a été créée pour mettre au point des outils d'analyse du trafic et de l'infrastructure d'Internet afin d'améliorer le réseau, de le rendre plus fiable, plus résistant et plus robuste.

Depuis 2000, un des projets publics phares de CAIDA a consisté à établir et à maintenir la carte du réseau des systèmes autonomes de base (le projet "Skitter"). Les plus récentes données de captures publiées représentent 12 517 grands systèmes autonomes, ce qui correspond à 1 134 634 adresses IP et à 2 434 073 liaisons (chemins logiques) entre elles.

Bien qu'elle semble étonnamment mystérieuse, la carte d'Internet établie par CAIDA a été créée en utilisant uniquement les données de configuration BGP des routeurs accessibles au public, les résultats de tests empiriques sur le réseau (traceroute) et les enregistrements WHOIS des blocs réseau. Cette carte est organisée en utilisant des coordonnées polaires. Les points représentant chaque système sont situés à un angle correspondant à l'emplacement physique du siège central d'un réseau et à un rayon correspondant à la "pertinence du peering" de ce système autonome. Ce dernier paramètre est obtenu en calculant le nombre de systèmes autonomes qui acceptent le trafic à partir de ce nœud particulier. Ainsi, les principaux systèmes sont situés vers le centre de la carte, tandis que les systèmes qui sont en contact direct avec seulement un ou deux nœuds sont situés près du périmètre extérieur. Les lignes dans le graphique correspondent tout simplement aux relations de peering entre les routeurs.

NOTE

Malheureusement, nous n'avons pas été autorisés à reproduire les graphiques du projet Skitter dans ce livre sans payer de droits. Je vous encourage cependant à voir cette magnifique photo en ligne à l'adresse suivante : http://www.caida.org/analysis/topology/as_core_network/pics/ascoreApr2003.gif, où elle est accessible gratuitement au grand public.

Une autre tentative notable d'établir la carte du réseau utilise une approche qui s'appuie sur l'analyse des distances observées entre les différents réseaux depuis un emplacement particulier (dans ce cas, depuis les laboratoires Bell) afin de construire une structure arborescente assez différente du maillage complexe créé par CAIDA. Cette analyse menée par Bill Cheswick en 2000[1] a abouti à la carte de la Figure 17.1. La structure de ce graphique ne dépend pas de l'emplacement physique ou administratif d'un système ; la distance relative d'un système par rapport au centre correspond au nombre de sauts entre ce nœud et les laboratoires Bell.

Bien que ces deux tentatives semblent impliquer la collecte et l'analyse d'un nombre important de données, il n'est pas trop difficile pour un amateur d'essayer de dresser la carte du réseau, même sur une liaison au débit assez faible. Découvrir tous les sous-réseaux routables publiquement avec un seul paquet pourrait ne nécessiter de générer que quelques gigaoctets de trafic – l'équivalent de quelques heures ou d'une journée avec une connexion DSL typique. Le seul risque est d'alerter certains administrateurs système, mais rares sont ceux qui ont un seuil de tolérance aussi bas vu la prolifération des vers informatiques et des attaques automatisées. Cartographier la structure observée d'Internet est possible et cela peut être enrichissant, surtout parce que cela peut nous en dire long sur l'organisation du réseau mondial.

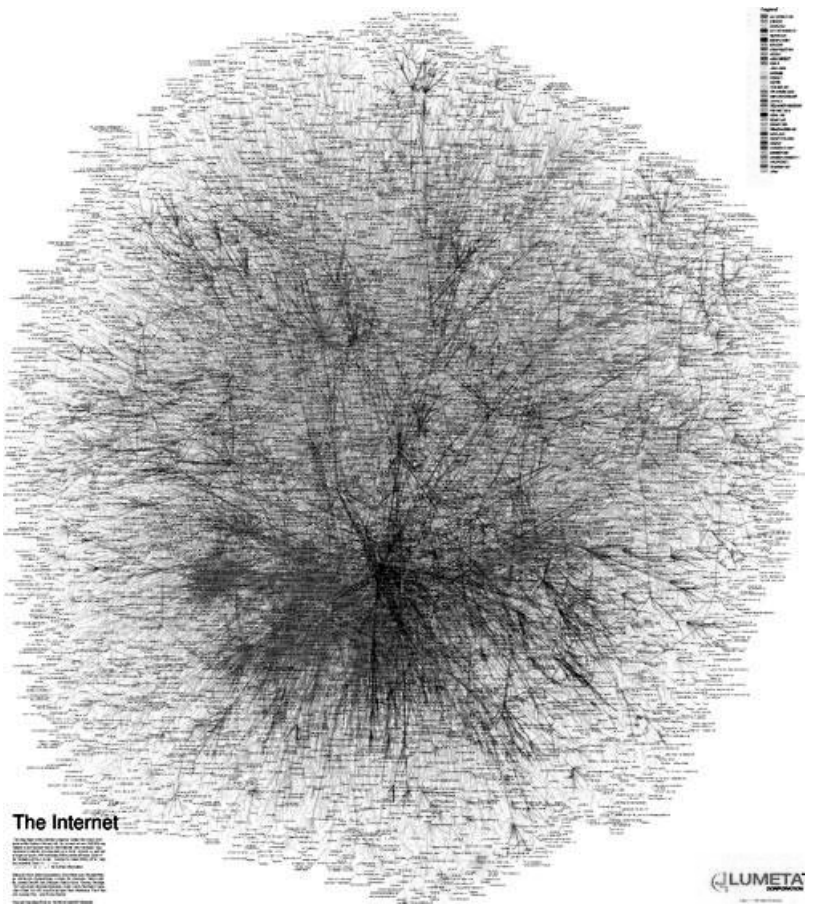


Figure 17.1

La carte d'Internet de Bill Cheswick.

Mais il s'avère que ces données, qu'il s'agisse des informations acquises par CAIDA, Bill Cheswick ou par n'importe quel utilisateur qui maîtrise le Net, peuvent également être utilisées afin de mieux comprendre la nature et l'origine d'un trafic mystérieux que nous pourrions un jour rencontrer.

Utiliser les données de topologie pour identifier l'origine du trafic

Le blind spoofing est un des problèmes majeurs d'Internet – ou du moins l'un de ses aspects les plus ennuyeux. Les paquets usurpés avec de fausses adresses sources ou des adresses spécialement choisies mais trompeuses peuvent être utilisés pour abuser la relation de confiance entre des ordinateurs, injecter du contenu malveillant (comme le mailing de masse non sollicité) sans laisser de traces concluantes ni d'information sur leur origine. Le blind spoofing peut également être utilisé pour masquer l'identité d'un attaquant qui scanne des systèmes (le "scanning leurre" évoqué au Chapitre 13). Mais l'usurpation utilisée pour réaliser des attaques par déni de service (DoS) est le pire fléau de tous.

Dans une attaque DoS typique, l'administrateur a une chance de voir l'origine du trafic malveillant dirigé contre un de ses services (vraisemblablement dans l'intention de provoquer une défaillance de ce service et de causer des désagréments ou d'entraîner des pertes pour l'opérateur). Toutefois, il est possible d'usurper de façon aléatoire les paquets malveillants. L'administrateur est alors sans défense et incapable de filtrer le trafic en provenance de l'attaquant sans déconnecter les autres utilisateurs. La seule situation dont il dispose consiste à collaborer avec le fournisseur en amont pour enquêter sur l'origine réelle du trafic sur la couche liaison et ensuite transmettre ces informations au FAI de l'attaquant, ce qui prend beaucoup de temps. Il faut également convaincre toutes les parties que l'affaire mérite qu'ils enquêtent (et y consacrent du temps et de l'argent) sans injonction judiciaire. Il est donc particulièrement important que l'administrateur système soit équipé d'outils et de méthodes pour distinguer le trafic usurpé du trafic légitime.

À l'époque où je vivais et travaillais aux États-Unis (je vis en Pologne actuellement), mon collègue Mark Loveless décida d'implémenter une idée initialement proposée par Donald McLachlan : il s'agissait de mesurer le temps de vie (TTL) sur le trafic réseau entre lui et l'expéditeur présumé d'un paquet pour déterminer automatiquement si un paquet entrant était usurpé. Il est important de pouvoir identifier l'origine d'un paquet sur le réseau dans un monde où l'information ne peut pas être digne de confiance. La capacité à le faire, ne serait-ce que dans certains cas, représenterait un grand avantage pour de nombreuses tâches analytiques et administratives, pour les raisons mentionnées précédemment.

Pour comprendre l'idée de Mark et de Donald, il faut considérer que le système distant à partir duquel nous observons le trafic se trouve à une distance logique spécifique par rapport à nous et que nous en sommes séparés par un certain nombre de périphériques réseau. Ainsi, tous les paquets légitimement envoyés par ce système montrent un certain TTL à l'arrivée, qui correspond à la valeur initiale du TTL configuré par défaut sur ce système moins le nombre de systèmes intermédiaires que le paquet a traversés (comme nous l'avons vu au Chapitre 9). En revanche, pour le trafic usurpé qui provient probablement d'un réseau complètement différent, ce TTL initial et cette distance sont très certainement différents de ce que l'observation mentionnée indique. L'utilitaire de Mark, despoof[2], compare le TTL observé du trafic spécialement initié avec celui précédemment reçu afin de différencier le trafic légitime du trafic usurpé.

Cependant, bien que cette méthode puisse parfaitement fonctionner dans des cas particuliers lorsqu'elle est utilisée contre des attaquants qui ne se doutent de rien, elle présente au moins deux inconvénients :

- Un attaquant paranoïaque peut mesurer les distances avant l'attaque et choisir un TTL qui corresponde à la valeur attendue. Bien que possible, cette astuce est assez difficile à implémenter. D'une part, l'attaquant peut être dans l'impossibilité matérielle de définir un TTL assez élevé pour obtenir une valeur qui corresponde à celle qui est attendue pour un paquet véritable une fois que le paquet arrive à destination. Ce plan de l'attaquant peut être mis en échec si le système dont il essaie d'usurper l'identité utilise un TTL par défaut égal ou proche de 255 (le maximum possible) et qu'il est plus éloigné de la cible que le système dont il usurpe l'identité (il lui est par conséquent quasi impossible d'envoyer un paquet qui ait le TTL voulu à son arrivée à destination). Bien sûr, peu de systèmes utilisent le TTL le plus élevé possible et il est rare qu'un attaquant souhaite usurper l'identité d'un système spécifique.
- L'attaquant peut également ne pas être en mesure de déterminer la distance exacte entre sa victime et le système usurpé s'il est loin d'eux, ne les connaît pas et ne connaît pas les spécificités de routage entre ces hôtes. Mais, si la victime utilise despoof pour implémenter dynamiquement des règles de filtrage et supprimer les paquets malicieux, l'attaquant peut simplement essayer différents TTL provenant de diverses sources jusqu'à ce qu'il constate que la victime n'est plus capable de faire la distinction (il serait évident de le constater : le système ciblé commencerait à montrer les signes d'une attaque réussie, notamment en termes de performances).
- Chaque fois qu'un paquet suspect est reçu, le destinataire doit ouvrir une enquête et attendre ensuite les résultats. Cela rend despoof impossible à utiliser comme base de défense automatique, surtout pour répondre aux attaques DoS. En revanche, cette méthode est encore très utile pour déterminer la véritable origine d'un "scan leurre".

Sans connaître la topologie d'un réseau, il est difficile de faire mieux avec despoof ; la technique d'analyse du TTL que cet utilitaire implémente est suffisamment bonne pour reconnaître et arrêter un grand nombre de découvertes et d'attaques individuelles, mais ensuite ?

Si l'on combine l'utilitaire de Mark avec des données en temps réel sur la structure du réseau et qu'on applique le fingerprinting passif pour déterminer le TTL initial d'un système qui envoie des requêtes spécifiques, cette technique devient alors beaucoup plus puissante. Ces données supplémentaires nous permettent de réaliser une première évaluation passive du trafic entrant, en comparant le TTL observé et le TTL initial avec la distance attendue qui est indiquée par la carte du réseau*. Comme la distance que nous devrions voir peut être déterminée sans initier aucune découverte active de la topologie du réseau, nous pouvons distinguer instantanément le trafic légitime du trafic malveillant sans grand effort. Cela rend ensuite possible de réagir à des incidents importants de façon assez fiable et de détecter des profils de sondage individuel sans alerter l'attaquant qu'un système de détection du spoofing est en place.

De toute évidence, il y a beaucoup à gagner à tenir compte de la structure d'un réseau dans le cas des relations de pair à pair. Mais la détection des usurpations n'en est qu'à ses débuts.

La triangulation du réseau à l'aide des données de la topologie maillée

La triangulation du réseau est une utilisation beaucoup plus intéressante des données de la topologie maillée du réseau pour analyser le trafic. On peut utiliser la triangulation du réseau pour déterminer l'emplacement approximatif d'un attaquant qui envoie des paquets usurpés sans l'aide du backbone de routage sous-jacent dès qu'il choisit d'attaquer plusieurs cibles simultanément ou successivement (ce qui nous représente une certaine consolation).

Pour être tout à fait exact, bien que la triangulation fonctionne mieux lorsque l'attaquant choisit plusieurs cibles, elle peut fonctionner assez bien dans certaines situations, même s'il choisit de n'attaquer qu'un seul service. Il est en particulier possible d'observer la même attaque à partir de différents points de vue lorsque l'objet attaqué a plusieurs adresses IP et que le service est fourni depuis plusieurs emplacements physiques afin de répartir la charge et de rendre l'ensemble de la structure tolérant aux pannes (ce qui est courant pour les services Web). Dans tous les autres cas de figure, on peut obtenir un

---

* La comparaison des TTL doit alors être effectuée avec une certaine marge d'erreur, car il peut y avoir plusieurs sauts supplémentaires dans les réseaux internes. De plus, certains chemins sont asymétriques et leur longueur peut varier légèrement en fonction de la direction dans laquelle le trafic est échangé.

ensemble de données sur une attaque lorsque les administrateurs système remarquent que plus d'un système est pris pour cible par un attaquant et qu'ils partagent leurs données au sujet de l'incident.

Quel que soit le cas de figure, une fois que les données censées provenir d'une seule source sont observées sur plus d'une destination, on peut trianguler. Pour chaque destination où le trafic est observé, seuls certains réseaux se trouvent à une distance qui corresponde à la distance observée pour le paquet incriminé (ce qui encore une fois peut être découvert en examinant le TTL*). En recoupant l'ensemble des réseaux possibles pour chaque point d'observation, on obtient un plus petit ensemble – ou bien souvent un seul réseau –d'où l'attaque peut provenir, comme illustré à la Figure 17.2.

La possibilité de suivre la trace du paquet par nous-mêmes nous rend indépendants sans condition des FAI et nous permet de localiser avec précision qui est en train d'attaquer ou de sonder notre réseau, voire peut-être de découvrir pourquoi.

Bien que cette approche soit beaucoup plus difficile à déjouer que le despoofing traditionnel, un attaquant habile peut encore être en mesure de tromper un observateur en utilisant aléatoirement un TTL (ou un ensemble de TTL) différent pour chaque cible. Bien sûr, il n'existe à notre connaissance aucun utilitaire capable de le faire à l'heure actuelle, mais cela pourrait changer.

La bataille est perdue ? Non, il existe un moyen d'empêcher les attaquants de nous tromper ainsi.

L'analyse de la saturation du réseau

Cette solution, baptisée "analyse de la saturation du réseau ", provient d'une étude présentée par Hal Brunch et Bill Cheswick à la conférence LISA de 2000[3]. Brunch et Cheswick proposèrent une utilisation intéressante des données topologiques arborescentes d'un réseau (semblables à celles du graphique de la Figure 17.1) obtenue à partir d'un endroit précis. Ils parvinrent à utiliser ces données pour détecter l'origine d'un type particulier de trafic usurpé : le déni de service. L'approche elle-même est assez simple et se fonde sur l'hypothèse selon laquelle une telle attaque saturerait non seulement le système contre lequel elle est menée mais aussi les routeurs intermédiaires. Cette saturation pourrait donc être mesurée de l'extérieur par la victime et utilisée pour (presque littéralement) remonter le long du fil en tirant dessus.

---

* Même si l'utilitaire de scan utilise des TTL aléatoires, il est possible d'évaluer la distance en utilisant le TTL maximal observé si un nombre de paquets peut être observé à chaque destination (ce qui est presque toujours le cas). Par exemple, si l'utilitaire de scan utilise aléatoirement un TTL initial situé dans une fourchette de 32 à 255 mais que, pour plusieurs milliers de paquets reçus à destination, aucun n'ait de TTL supérieur à 247, l'hôte a de fortes chances de se trouver à 255 - 247 = 8 systèmes de distance.

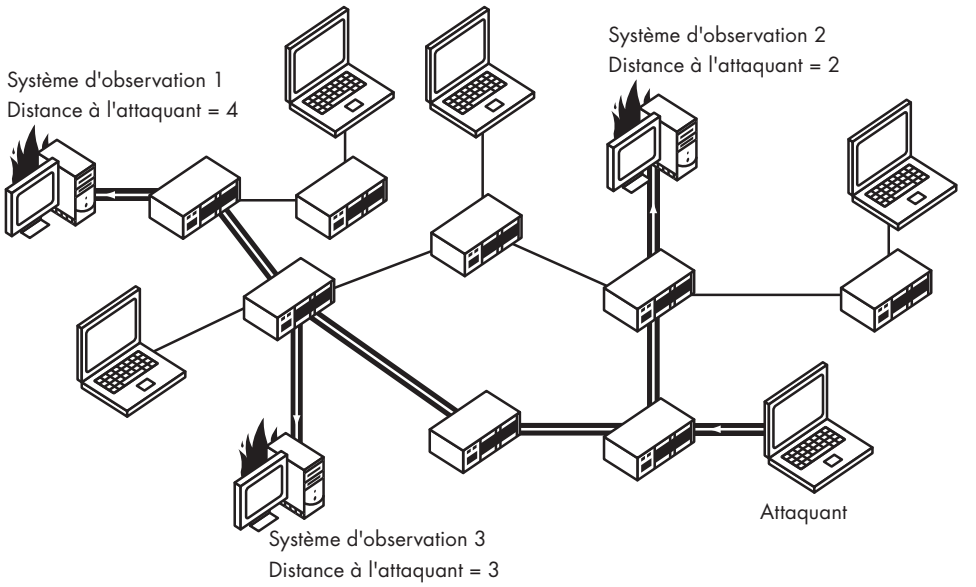


Figure 17.2

Une triangulation du réseau simple : une seule origine est conforme à toutes les observations. L'attaquant peut usurper des adresses sources, mais il ne peut tromper les victimes.

Pour tester la saturation des liaisons du réseau, on commence par créer ou obtenir une arborescence de liaisons à partir de l'endroit où l'on se trouve pour tous les réseaux Internet, puis on passe de branche en branche dans cette structure arborescente quand une attaque se produit. Pour chaque branche (qui, en réalité, désigne une connexion à un routeur supérieur dans la hiérarchie), on peut mesurer par itération la charge réseau sur le nœud en envoyant du trafic de test vers ou à travers le routeur qui lui est associé (dans ce document, une charge UDP [*User Datagram Protocol*] est utilisée, mais les requêtes ICMP ou tout autre type de message peuvent être également utilisés). On choisit un des nœuds les plus saturés comme candidat potentiel pour le trafic entrant, puis on établit la liste et on teste toutes les branches qui partent de ce nœud jusqu'à remonter à l'origine du trafic.

La Figure 17.3 illustre un simple cas de retraçage. Dans la première phase, le système attaqué tente de mesurer les performances des trois routeurs Internet les plus proches quand une attaque se produit et conclut que le premier routeur (en haut) est le plus saturé.

À partir de ces informations, la victime choisit de tester uniquement les routeurs connectés directement (*peering*) à ce périphérique. Dans ce cas de figure, seulement trois périphériques sont testés (les six autres ne le sont pas car ils n'ont pas de liaison de pair à

pair avec ce périphérique) et, de nouveau, le premier est le plus saturé. Ce processus continue jusqu'à ce qu'un routeur directement relié à un réseau sur lequel l'emplacement physique et le propriétaire sont disponibles dans les bases de données publiques corresponde au système d'extrémité.

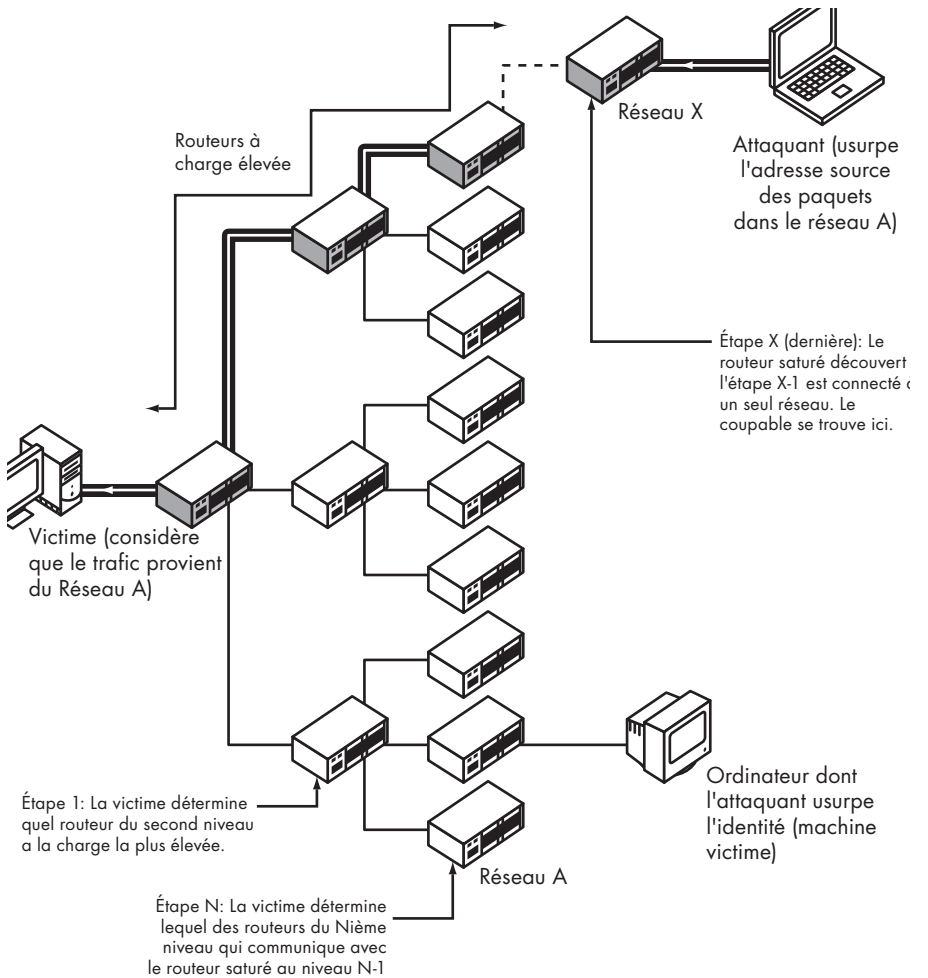

Réseau X
Attaquant (usurpe l'adresse source des paquets dans le réseau A)
Routeurs à charge élevée
Étape X (dernière): Le routeur saturé découvert l'étape X-1 est connecté
un seul réseau. Le coupable se trouve ici.
Victime (considère que le trafic provient du Réseau A)
Ordinateur dont l'attaquant usurpe l'identité (machine victime)
Étape 1: La victime détermine quel routeur du second niveau a la charge la plus élevée.
Réseau A
Étape N: La victime détermine lequel des routeurs du Nième niveau qui communique avec le routeur saturé au niveau N-1

Figure 17.3

Une attaque retracée de façon récursive en utilisant des données sur la topologie du réseau et des tests de saturation.

Un problème potentiel se pose : certains périphériques peuvent être saturés pour des raisons autres que la gestion du trafic DoS tandis que d'autres peuvent avoir beaucoup de cycles processeur libres et donc ne pas être considérablement affectés lorsqu'ils relaient du trafic malveillant.

Pour résoudre ce problème, l'étude propose de mettre artificiellement une charge à court terme sur le routeur (en générant un trafic supplémentaire), puis d'observer la manière dont ce test influe sur la bande passante et la latence des requêtes DoS ; si ce périphérique relaie effectivement des paquets malveillants, le taux de l'attaque doit baisser lorsque nous augmentons sa charge (encore une fois, en générant des requêtes fictives TCP, UDP ou ICMP supplémentaires qui visent plus à consommer la puissance du processeur d'un périphérique qu'à congestionner ses interfaces). Par conséquent, il devrait y avoir une corrélation uniquement sur les branches qui sont impliquées dans la fourniture du trafic malveillant.

Cette méthode brillante et simple a été utilisée avec succès dans des environnements tests. En revanche, comme elle implique d'interagir avec les routeurs et d'augmenter leur charge, certaines considérations éthiques entrent en jeu lorsque l'on envisage de l'utiliser dans le monde réel.

Matière à réflexion

La principale difficulté qu'il y a à utiliser les techniques présentées dans ce chapitre pour traquer les assaillants est que nous devons construire et mettre à jour les cartes du réseau pour chaque emplacement. Il n'est pas très évident de savoir à quelle fréquence ces cartes devraient être mises à jour ni quelles méthodes se révéleraient les plus fiables et les moins intrusives.

Une autre question tient au fait que la majeure partie de l'infrastructure de base d'Internet est redondante. Certains chemins de rechange peuvent n'être choisis qu'en cas de panne ou de saturation du chemin principal, mais cette commutation se produira dans certains cas pour équilibrer la charge. Ainsi, certaines cartes empiriques risquent de devenir obsolètes en l'espace de quelques minutes ou de quelques heures, bien que de tels cas ne soient pas très fréquents.

En fin de compte, même si l'utilisation individuelle et privée de différentes tactiques de despoofing se révèle très fructueuse, de nombreuses questions subsistent avant de pouvoir déployer ces techniques à grande échelle. Et certaines questions ne relèvent pas uniquement du domaine technique.

18

En regardant le vide

Quand on baisse les yeux vers l'abîme, ce qui ne nous tue pas nous rend plus forts.

Nous avons examiné de nombreuses façons de découvrir des informations et d'intercepter des données en observant les communications entre deux systèmes ou en observant les effets secondaires de ces communications. Cependant, l'histoire ne s'arrête pas là. Parfois, en détournant les yeux de la cible que nous essayons de découvrir, nous pouvons voir encore plus de choses.

Tout un ensemble de méthodes, couramment baptisé "surveillance des trous noirs", est dédié à l'observation et à l'analyse du trafic indésirable ou non sollicité qui arrive accidentellement, à tort ou dans une forme altérée à une destination spécifique. Ces méthodes consistent le plus souvent à simplement exécuter un utilitaire de suppression des paquets, puis à soigneusement analyser et à émettre des théories sur toutes les occurrences.

Bien que dans l'idéal nous n'ayons rien à gagner à rechercher des données là où nous ne sommes pas censés les trouver, nous pouvons en fait utiliser ces méthodes pour recueillir de nombreuses informations et des indices précieux sur l'état d'un réseau dans son ensemble. Même si ces informations sont le plus souvent aléatoires et que nous ne pouvons pas décider qui nous écoutons, nous pouvons tout de même en tirer profit.

Les tactiques d'observation directe

La surveillance des trous noirs permet de détecter et d'analyser les tendances mondiales d'attaques. Beaucoup de pirates en possession de nouvelles techniques d'attaques scannent souvent de gros blocs d'adresses réseau pour trouver des cibles vulnérables qui peuvent être compromises et finalement utilisées pour des activités illicites (sans doute pour créer des machines zombies ou créer des botnet afin de réaliser des attaque automatisées). Nous pouvons utiliser la surveillance des trous noirs pour nous alerter que de nouvelles vulnérabilités sont exploitées, en observant simplement l'augmentation des scanners de vulnérabilité effectués à partir de diverses sources.

De nombreux administrateurs réseau déploient une surveillance des trous noirs. Ils la combinent parfois avec la technique du pot de miel (dans laquelle un système sur le réseau sert "d'appât" pour attirer des attaquants, intercepter leurs utilitaires et identifier leurs techniques[1]) afin d'obtenir un système d'alerte qui leur permette d'être les premiers à connaître les nouveaux types de vers et d'autres programmes malveillants (vous pouvez également utiliser le trafic du trou noir pour calibrer les "niveaux de bruit" et détecter les attaques ciblées contre vos serveurs plus efficacement sans tenir compte des activités malveillantes automatisées et réalisées à l'aveugle).

Des chercheurs comme Dug Song et Jose Nazario (Jose, dans son livre *Defense and Detection Strategies against Internet Worms*[2]) ont tenté d'analyser l'activité du trou noir au cours des épidémies massives de vers sur un réseau. Leur objectif est de mieux comprendre et de modéliser la dynamique de répartition du réseau (propagation initiale et réinfection) et de tester l'efficacité et la persistance des algorithmes d'infection des vers. Leurs recherches nous aideront à concevoir les futures défenses contre les menaces distribuées à grande échelle, tout en apportant un éclairage précieux sur l'état du réseau actuel. Quelques exemples de leurs conclusions sont présentés aux Figures 18.1 à 18.4.

La Figure 18.1 montre comment un ver se propage lors d'une épidémie. Les données sont fondées sur le nombre de tentatives d'attaque observées sur le port TCP 137, une partie de l'implémentation NetBIOS de Windows installée par défaut sur tous les ordinateurs sous Windows et qui est la cible de nombreux types de logiciels malveillants se propageant par eux-mêmes. Comme vous pouvez le remarquer à cette figure, après une semaine de propagation initiale durant laquelle le nombre de sites infectés (sources) et de systèmes attaqués sur le réseau trou noir observé a constamment et rapidement augmenté, une soudaine période de stabilisation suit qui s'étend sur plus d'un mois et compte des crêtes et des creux importants. Une telle empreinte de propagation est typique d'un ver et des conditions du réseau dans lesquelles il opère ; elle reflète également des détails sur la sélection de la cible et sur les algorithmes d'infection utilisés.

La Figure 18.2 montre un autre aspect de l'algorithme de propagation du ver et dépeint les propriétés de l'algorithme de sélection des cibles. Dans ce cas, un ver populaire qui cible des serveurs Microsoft SQL semble avoir une couverture assez continue de l'espace d'adressage (bien que les adresses ayant des octets entre 200 et 225 environ soient nettement plus souvent choisies, et le ver semble sauter les valeurs supérieures à 225).

La Figure 18.3 montre le même graphique pour un autre ver de réseau, Slapper. Ce ver cible les systèmes Linux en exploitant une faille dans une librairie de cryptage OpenSSL populaire. L'algorithme semble offrir une couverture beaucoup plus uniforme mais beaucoup moins continue et compte des trous béants pour certaines valeurs.

La Figure 18.4 montre la persistance des motifs du ver dans le temps. Par exemple, certains vers semblent mourir lorsque les systèmes sont corrigés et désinfectés, tandis que d'autres utilisent des algorithmes qui entraînent des augmentations et des diminutions soudaines et récurrentes (ce motif est familier pour quiconque a étudié des modèles épidémiologiques ou de population fondés sur des phénomènes naturels).

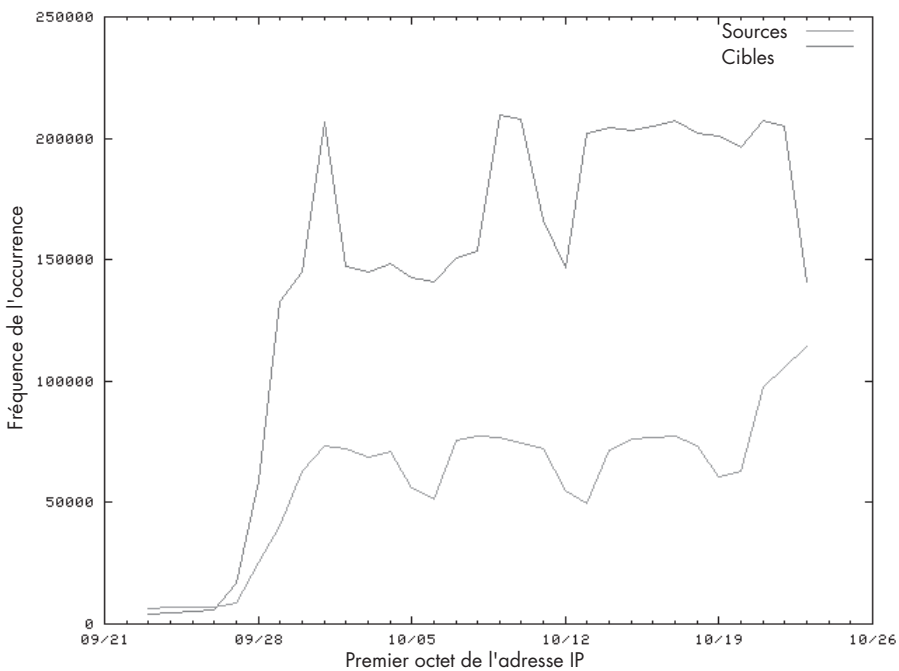


Figure 18.1

Les caractéristiques de propagation du ver sous Windows.

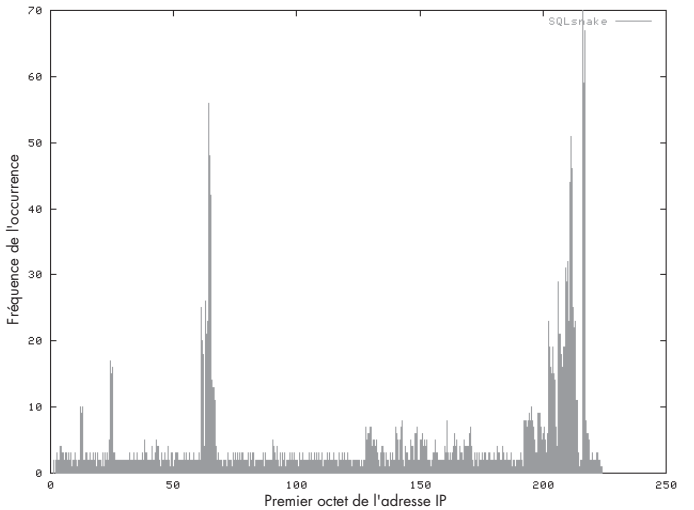


Figure 18.2

L'histogramme de l'algorithme de sélection de cible du ver SQLSnake ; notez la couverture non uniforme mais généralement continue de l'espace d'adressage.

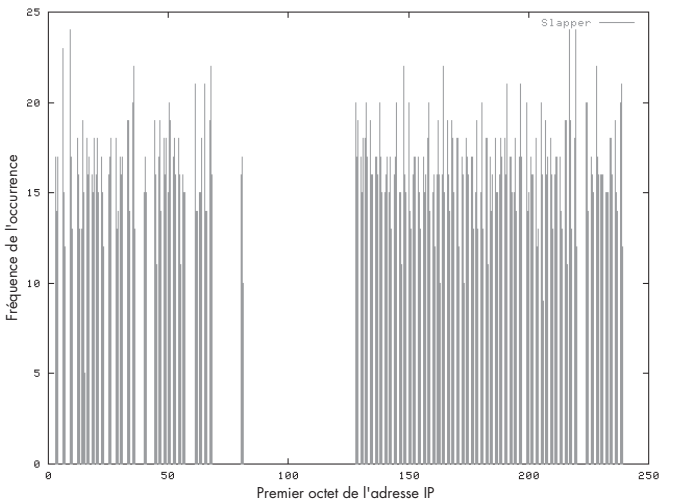


Figure 18.3

L'histogramme de l'algorithme de sélection de cible du ver Slapper montre une répartition beaucoup plus homogène, mais non continue. Les trous indiquent que les bits les moins importants de chacune des adresses "aléatoires" sont constants, peut-être en raison d'une erreur de programmation.

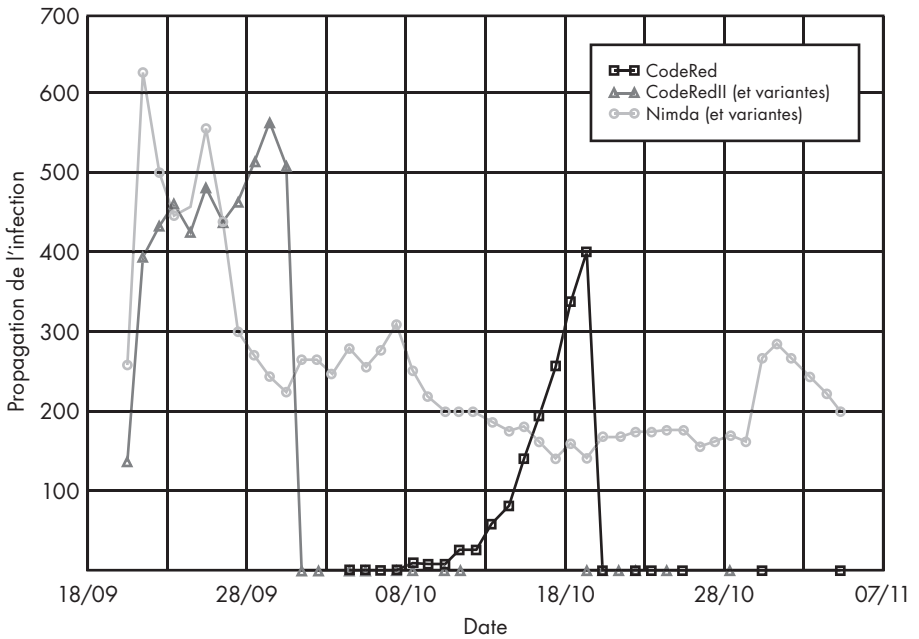


Figure 18.4

La persistance du ver dans le temps. Notez que la complexité du schéma de propagation du ver CodeRed, qui se comporte comme un modèle de population biologique.

Comme Jose et ses collègues s'efforcent de le démontrer, la surveillance des trous noirs peut ne pas être seulement une activité de routine complètement inutile mais être également un excellent moyen de découvrir la vie secrète de toutes les choses malveillantes. Hélas, les choses ne s'arrêtent pas là ! En n'observant que le trafic que nous estimons nous viser, nous manquons les bits de données les plus intéressants.

L'analyse des retombées du trafic de l'attaque

L'autre application de la surveillance des trous noirs s'appuie sur l'observation du trafic qui ne nous a jamais été destiné mais qui est simplement un effet secondaire d'une autre activité.

Ici, nous pouvons voir comment un certain nombre de systèmes de reconnaissance et d'attaque utilisent l'usurpation d'adresse pour dissimuler l'identité d'un attaquant. On peut estimer qu'un administrateur aura quelque difficulté à différencier le trafic leurre des adresses usurpées réellement utilisées par l'attaquant. Même si, comme je l'ai indiqué

dans les chapitres précédents, cette approche ne garantit pas l'anonymat complet de l'attaquant, un administrateur doit pour réussir le "despoofing" du trafic implémenter une vaste journalisation et des mesures additionnelles au moment de l'attaque. Comme ces procédures ne sont pas toujours implémentées, les attaquants peuvent souvent réaliser efficacement leurs attaques par usurpation et rester dans l'ombre.

Que des paquets soient usurpés ou non, le système attaqué répondra en toute bonne foi à toutes les requêtes, y compris celles en provenance d'adresses créées de toutes pièces. Cependant, seules les réponses aux paquets dont l'adresse source est correcte parviendront en retour à l'expéditeur ; toutes les autres sondes génèrent d'autres réponses qui s'éparpillent dans tout le réseau Internet. Et on peut souvent les capturer.

Bien qu'il semble peu probable de recevoir un paquet aussi mal acheminé, n'oubliez pas qu'un nombre considérable de paquets SYN+ACK, RST+ACK et RST sont générés en réponse à des scans leurres ou à des attaques SYN en grande quantité. L'espace d'adressage Internet est très vaste et des millions de paquets sont généralement impliqués dans ces attaques, mais il est fort probable qu'avec le temps certains atteignent chaque bloc réseau. Bien qu'il n'y ait qu'une chance sur 4 294 967 296 (1 jusqu'à 2^{32}) qu'un seul paquet usurpé généré aléatoirement rebondisse vers une adresse spécifique, cette probabilité passe de 1 sur 16 777 216 (1 jusqu'à 2^{24}) si l'on considère qu'un petit sous-réseau affecté à une petite entreprise ou à une organisation est habituellement composé de 256 adresses (réseau de classe C ou équivalent). Cette probabilité peut encore être améliorée en excluant les plages d'adresses qui sont connues pour être réservées à des fins spéciales ou celles qui ne méritent pas qu'on en tienne compte et sont donc exclues dans certains types d'attaques.

Comme le poids d'un seul paquet SYN est d'environ 40 octets (et se compresse bien en vrac) et qu'une liaison réseau typique à la disposition d'un attaquant occasionnel a un débit d'environ 10 à 150 kilo-octets par couche IP par seconde (DSL bas de gamme et liaison T1, respectivement), il peut envoyer 250 à près de 3 000 paquets dans ce délai – ou de 900 000 à environ 10 millions de paquets par heure*.

Pour qu'une attaque DoS typique produise des résultats visibles et entraîne des inconvénients majeurs pour la victime, elle doit en général être menée pendant plusieurs heures ou plusieurs jours (l'attaquant veut gêner sa victime aussi longtemps que possible). De ce fait, des dizaines, voire des centaines, de millions de paquets sont envoyés, ce qui génère un nombre semblable de réponses SYN+ACK ou RST+ACK.

En raison de cette énorme quantité de trafic, on peut s'attendre à ce que même une entité relativement petite constate les retombées d'une petite attaque SYN flood en cours

---

* Notez que les attaquants chevronnés déterminés et compétents dans les attaques par dénis de service disposent souvent de dizaines ou de centaines de nœuds "zombies" à leurs ordres, ce qui augmente de façon spectaculaire cette estimation.

d'exécution, même si le destinataire hôte abandonne de nombreux paquets de l'attaque. En outre, les administrateurs en mesure de surveiller des réseaux de classe B (65 356 adresses, le plus souvent la propriété de grandes entreprises, des fournisseurs d'accès à Internet, des instituts de recherche, etc.) seront capables de détecter rapidement des événements beaucoup moins importants.

Puisque toutes les réponses retombées dans une attaque DoS contiennent certains détails des messages fabriqués par l'attaquant pour déclencher ces réponses (comme les numéros de port et les numéros de séquence, les informations dans le temps, et ainsi de suite), on peut utiliser ces réponses pour extraire des informations importantes sur le type de l'attaque et sur son ampleur. Nous pouvons utiliser ces réponses pour déterminer si un service spécifique a été pris pour cible, le nombre de systèmes ciblés, la bande passante dont dispose l'attaquant, et l'utilitaire utilisé pour effectuer l'attaque (en examinant la sélection du port source, le choix des numéros de séquence et les motifs IP "aléatoires"*).

Enfin, en analysant les sources de ces réponses en ricochet, on pourrait remarquer que tel ou tel segment de réseau est attaqué ou est en mesure d'identifier les "tendances hostiles" globales et ainsi peut-être mieux se préparer si une industrie ou une entreprise en particulier est la cible. Nous pouvons également utiliser ces informations pour en apprendre davantage sur les attaques qui sont masquées par la victime ou pour identifier les fausses allégations d'attaques (parfois, les chargés de communication affirment avoir été l'objet d'attaques par des cyberterroristes pour justifier des pertes financières ou pour des raisons politiques. Ces dernières années, certains experts ont accusé le groupe SCO d'avoir mis ses serveurs hors ligne en prétendant être victime d'une attaque DoS coordonnée pour discréditer la communauté des utilisateurs de Linux).

Détecter les données malformées ou mal dirigées

Cette application de la surveillance des trous noirs repose sur le contrôle du trafic qui ne semble pas avoir de sens mais qui parvient toujours à une destination précise. Pour mieux illustrer ce problème, permettez-moi la digression suivante.

En 1999, avec l'aide d'un groupe d'amis et de collègues en Pologne, j'ai commencé un modeste projet après mes heures de travail. Nous cherchions à suivre la trace d'une série de paquets RST+ACK difficiles à expliquer dont nous avions constaté la présence dans les réseaux sous notre surveillance et à surveiller les tendances du trafic inhabituel et non sollicité arrivant sur des segments inutilisés du réseau en général. Ce fut très amusant et, comme vous pouvez l'imaginer, nous avons échafaudé de nombreuses hypothèses pour

---

* Par exemple, certains utilitaires "usurpent" uniquement les paquets qui proviennent d'adresses IP paires ou impaires en raison de défaillances dans leur code. Des analyses semblables à celles menées par Jose Nazario se révèlent tout aussi capables d'identifier les utilitaires d'attaque que les vers.

tenter d'expliquer certains des cas les plus insolites. Notre recherche nous a aussi permis d'en apprendre plus sur le monde qui nous entoure car nous avons rencontré du trafic extrêmement bizarre et apparemment inexplicable qui, une fois correctement analysé, nous donna un meilleur aperçu des grandes conspirations de notre monde électronique.

Bien qu'officiellement abandonné, ce projet s'est retrouvé dans mon "musée privé des paquets cassés"[3] sur ma page Web semi-humoristique consacrée à la traque, à la documentation et à l'explication des paquets qui ne devraient jamais avoir atteint leur destination ou qui ne devraient jamais avoir l'aspect qu'ils ont. L'objectif déclaré de ce musée est le suivant :

> Il s'agit de fournir un abri pour les paquets étranges, indésirables, malformés, abandonnés et voués à être des phénomènes de foire, alors que nous, simples mortels, les rencontrons sur les chemins tortueux de la vie. Les pièces de notre collection ou ses habitants, si vous préférez, ne sont souvent que l'ombre de ce qu'ils étaient avant qu'ils ne rencontrent un routeur hostile et défaillant. Certains d'entre eux sont malformés depuis leur naissance dans les profondeurs d'une mauvaise implémentation de pile IP. D'autres paquets étaient normaux, tout comme leurs amis (vous ou moi), mais se perdirent en recherchant le sens ultime de leur existence et arrivèrent où ils n'auraient jamais dû. Quoi qu'il en soit, nous essayons de découvrir l'histoire singulière de la vie de chaque paquet et de vous aider à comprendre à quel point il est difficile d'être un messager solitaire dans l'univers hostile des bits et des octets.

Et c'est à ce dernier type que la surveillance des trous noirs se résume. Même si cette tâche peut paraître inutile au premier abord, il est illusoire de penser qu'elle l'est. Ce musée a permis la découverte passive de sombres secrets sur les différents périphériques propriétaires et de réseaux bien protégés, et exécuter une expérience similaire ailleurs ne manquerait pas de fournir des résultats identiques ou supérieurs.

Mon musée contient certaines merveilles comme les suivantes :

- Des paquets provenant de réseaux avec un type précis d'accélérateur Web, de routeur ou de pare-feu ; le périphérique ajoute, découpe ou tronque certaines des données. Une faille dans plusieurs périphériques de Nortel CVX qui est responsable de l'altération occasionnelle des en-têtes de paquets TCP (comme nous l'avons vu au Chapitre 11) constitue un bon exemple. Le caractère unique de cette faille nous permet d'apprendre beaucoup sur un certain nombre de réseaux distants sans avoir à réellement les sonder.
- Plusieurs bruits de liaisons entraînent la création de paquets ne contenant que des données inutiles ou qui n'appartiennent sans doute pas à une connexion spécifique. L'une des pièces les plus surprenantes est un trafic non sollicité contenant des données qui semblent provenir d'un vidage du contenu de la zone DNS .de (une liste de tous les domaines de l'Allemagne). Ce trafic ne peut pas être né n'importe où, puisque de simples mortels n'ont pas le droit d'obtenir une telle liste. Ces données

doivent plutôt provenir d'une partie autorisée en mesure de les obtenir et de les transférer et doivent avoir été tronquées soit par l'expéditeur soit par un périphérique en cours de route. Bien que tous ces cas n'apportent que peu d'éclaircissement sur la nature des accidents de parcours sur le réseau, des cas comme celui-ci constituent des découvertes inattendues mais précieuses qui enrichissent l'observateur.

On pourrait également citer les cas d'espionnage apparents camouflés en trafic régulier et les nombreux autres hoquets de codage ou des réseaux. Mais assez fanfaronné, si vous avez envie d'en savoir plus, visitez l'adresse suivante : http://lcamtuf.coredump.cx/mobp/.

Matière à réflexion

Beaucoup considèrent la surveillance des trous noirs comme un autre moyen de détecter les attaques contre leurs systèmes (et peut-être un moyen coûteux, étant donné la rareté des ressources publiques sur l'espace IP). Mais la valeur réelle de cette technique est qu'elle permet non seulement d'identifier les attaques connues (quelque chose qui peut être fait tout aussi bien dans de nombreux autres endroits, sans gaspiller l'espace IP) mais aussi de détecter et d'analyser les motifs subtils qui seraient autrement noyés sous le "niveau de bruit" dans un réseau largement utilisé.

Naturellement, effectuer ce type de contrôle type des trous noirs n'est pas facile et reste coûteux. Il faut du temps pour savoir comment trouver cette aiguille dans la botte de foin de l'activité du ver ou du pirate. Et elle n'a généralement aucune signification dans un assez vaste réseau, à part pour établir des statistiques.

Pourtant, la joie de trouver enfin l'aiguille mérite souvent qu'on essaie.

Épilogue

Dans lequel le livre approche de sa conclusion.

Ce livre se termine, mais j'espère que votre voyage commence. J'ai pris un grand plaisir à vous guider à travers le monde des problèmes de sécurité complexes et inhabituels que j'apprécie le plus, et j'espère que c'est également votre cas. Que vous soyez un professionnel de la sécurité chevronné – peut-être plus expérimenté et compétent que moi – ou que vous découvriez avec enthousiasme ce domaine, j'espère vous avoir ouvert de nouvelles perspectives et que vous considérez maintenant la sécurité comme un défi et un art et non pas comme un ensemble d'obstacles qui doivent être éliminés ou contournés.

En comprenant les relations subtiles entre des composants et processus sans lien apparent, on peut réellement lutter contre les problèmes de sécurité les plus dangereux et difficiles mais aussi évaluer et atténuer efficacement les risques quotidiens. Les problèmes de sécurité doivent être considérés comme une fonction d'une solution à presque tous les défis dans le monde de l'informatique, quelle que soit sa simplicité ou sa portée limitée, et non pas comme des circonstances empêchant de faire des affaires. Ce n'est qu'en voyant la magie et le charme des univers complémentaires et la façon subtile avec laquelle ils interagissent que nous pouvons éviter la routine et commencer à vraiment profiter de notre travail ou à comprendre notre passe-temps.

Mais ce n'est ni le moment ni le lieu de jouer sur la corde sensible.

Merci de m'avoir accompagné.

Notes bibliographiques

Chapitre 1

1. Alan Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem", Proceedings of the London Mathematical Society, Series 2, 42 (1936).
2. R. L. Rivest, A. Shamir et L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Massachusetts Institute of Technology (1978).
3. Ueli M. Maurer, "Fast Generation of Prime Numbers and Secure Public-Key Cryptographic Parameters", Institute for Theoretical Computer Science, ETH Zurich, Suisse (1994).
4. Donald E. Knuth, *The Art of Computer Programming*, Volume 2 : Seminumerical Algorithms, 3e ed. Addison-Wesley (1997).
5. H. Krawczyk, “How to Predict Congruential Generators”, Journal of Algorithms 13, n° 4 (1992).
6. S. Bakhtiari, R. Safavi-Naini et J. Pieprzyk, "Cryptographic Hash Functions : A Survey", Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Australie (1995).
7. Dawn Xiaodong Song, David Wagner et Xuqing Tian, "Timing Analysis of Keystrokes and Timing Attacks on SSH", University of California, Berkeley (2001).

8. Claude E. Shannon, “Prediction and Entropy of Printed English", Bell Systems Technical Journal 3 (1950).
9. Benjamin Jun, Paul Kocher, "The Intel Random Number Generator", Cryptography Research Inc. (1999).
10. "Evaluation of VIA C3 Nehemiah Random Number Generator", Cryptography Research Inc. (2003).
11. Michael A. Hogye, Christopher T. Hughes, Joshua M. Sarfaty et Joseph D.Wolf, "Analysis of Feasibility of Keystroke Timing Attacks Over SSHConnections", CS588 Research Project, School of Engineering and Applied Science, University of Virginia (2001).

Chapitre 2

1. Yurii Rogozhin, “A Universal Turing Machine with 22 States and 2 Symbols”, *Romanian Journal of Information Science and Technology*, 1, n° 3 (1998).
2. Aleksandar Milenkovic et Jeffrey Kulick, “Demystifying Intel Branch Predictors”, Electrical and Computer Engineering Department, University of Alabama, Huntsville (2002).
3. Paul C. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”, Cryptography Research Inc. (1999).
4. *Intel 80386 Programmer’s Reference Manual*, section 7.2.IMUL, Intel Corp. (1986).
5. E. Biham et A. Shamir, “Differential Fault Analysis: Identifying the Structure of Unknown Ciphers Sealed in Tamper-Proof Devices” (1996).

Chapitre 3

1. Wim van Eck, “Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk?”, PTT Laboratories, Pays-Bas (1985).
2. Ian A. Murphy, “Who’s Listening?”, IAM/Secure Data Systems (1988, 1997).
3. Winn Schwartau, *Information Warfare*, 2e ed., Thunder’s Mouth Press, New York (1996).

Chapitre 5

1. John A.C. Bingham, *The Theory and Practice of Modem Design*, Wiley-Interscience (1988).
2. Electronic Industries Association, Engineering Department, “Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange” (1991).

3. Charles E. Spurgeon, *Ethernet: The Definitive Guide*, O'Reilly and Associates (2000).
4. Joe Lughry et David A. Umphress, "Information Leakage from Optical Emanations", ACM Trans. Info. Sys. Security 5, n° 3 (2002).
5. Adi Shamir et Eran Tromer, "Acoustic Cryptanalysis: On Nosy People and Noisy Machines". Une présentation en anglais de cet ouvrage est disponible à l'adresse suivante : http://www.wisdom.weizmann.ac.il/~tromer/acoustic/ (2004).
6. Paul Kocher, Joshua Jaffe et Benjamin Jun, "Differential Power Analysis", Cryptography Research, Inc. (2000).

Chapitre 6

1. J. Postel et J. Reynolds, "RFC-1042: A Standard for the Transit of Internet Protocol Datagrams Over IEEE 802 Networks", Network Working Group (1988). http://www.ietf.org/rfc/rfc1042.txt.
2. Ofir Arkin et Josh Anderson, "EtherLeak – Ethernet Frame Padding Information Leaks", @Stake (2003). http://www.atstake.com/research/advisories/2003/atstake_etherleak_report.pdf.

Chapitre 7

1. David C. Plummer, RFC 826, "An Ethernet Address Resolution Protocol", Network Working Group (1982).
2. Louis Senecal, "Layer 2 Attacks and Their Mitigation", Cisco (2002).

Chapitre 8

1. J. Case, M. Fedor, M. Schoffstall et J. Davin, RFC 1157, "A Simple Network Management Protocol", Network Working Group (1990).
2. Institut für Bankinnovation GmbH, "PSYLock: a typing behaviour based psychometrical authentication method" (2003). http://pc50461.uni-regensburg.de/ibi/de/leistungen/research/projekte/einzelprojekte/ psylock_english.htm.
3. Solar Designer et Dug Song, "Passive Analysis of SSH (Secure Shell) Traffic", Openwall Project (2001). http://www.openwall.com/advisories/OW-003-ssh-traffic-analysis.
4. Nikita Borisov, Ian Goldberg et David Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11" (2001).

Chapitre 9

1. J. Postel, université de Californie du Sud, "RFC 791: Internet Protocol", Network Working Group (1981).
2. J. Postel, université de Californie du Sud, "RFC 796: Address Mappings", Network Working Group (1981).
3. J. Mogul et S. Dearing, "RFC 1191: Path MTU Discovery", Network Working Group (1990).
4. J. Postel, université de Californie du Sud, "RFC 768: User Datagram Protocol", Network Working Group (1980).
5. J. Postel, université de Californie du Sud, "RFC 793: Transmission Control Protocol", Network Working Group (1981).
6. S. Bellovin, "RFC1948: Defending Against Sequence Number Attacks", Network Working Group (1996).
7. V. Jacobson et B. Braden, "RFC1232: TCP Extensions for High Performance", Network Working Group (1992).
8. M. Mathis, J. Mahdavi, S. Floyd et A. Romanow, "RFC2018: TCP Selective Acknowledgment Options", Network Working Group (1996).
9. B. Braden, "RFC1644: T/TCP – TCP Extensions for Transactions – Functional Specification", Network Working Group (1994).
10. J. Postel, université de Californie du Sud, "RFC 792: Internet Control Message Protocol", Network Working Group (1981).
11. Lance Spitzner, *Honeypots: Tracking Hackers*, éditions Addison-Wesley (2002).
12. R. Morris, "A Weakness in the 4.2BSD UNIX TCP/IP Software", Laboratoires AT&T Bell (1985).

Chapitre 10

1. Michal Zalewski, "Strange Attractors and TCP/IP Sequence Number Analysis", BindView Corporation (2001). http://www.bindview.com/Support/RAZOR/Papers/2001/
2. S. Bellovin, "Defending Against Sequence Number Attacks", Network-Working Group (1996). http://www.ietf.org/rfc/rfc1948.txt.
3. Joe Stewart, "DNS Cache Poisoning: the Next Generation" (2002). http://www.lurhq.com/dnscache.pdf.

Chapitre 11

1. Elizabeth D. Zwicky, Simon Cooper et D. Brent Chapman, *Building Internet Firewalls*, O'Reilly & Associates (2000).
2. G. Ziemba, D. Reed et P. Traina, "RFC1858: Security Considerations for IP Fragment Filtering", Network Working Group (1995).
3. Uriel Maimon, "TCP Port Stealth Scanning", *Phrack Magazine* n° 49 (1996).
4. J. Postel et J. Reynolds, "RFC959 : File Transfer Protocol", Network Working Group (1985).
5. Mikael Olson, "Extending the FTP ALG Vulnerability to any FTP client", liste de diffusion VULN-DEV (2000). http://www.securityfocus.com/archive/82/50226.
6. Michal Zalewski, "Linux Kernel IP Masquerading Vulnerability", Bindview Corporation (2001). http://razor.bindview.com/publish/advisories/adv_LkIPmasq.html.
7. R. Braden (edition), "RFC1122: Requirements for Internet Hosts—Communication Layers", Network Working Group (1989).

Chapitre 13

1. Salvatore Sanfilippo, "New TCP Scan Method". Bugtraq (1998). http://seclists.org/bugtraq/1998/Dec/0082.html.

Chapitre 14

1. World Wide Web Consortium. http://www.w3c.org/History.html.
2. Vannevar Bush, "As We May Think", *Atlantic Monthly* 176, n° 1, p 101-108 (1945).
3. Tim Berners-Lee, "Basic HTTP". http://www.w3c.org/Protocols/ HTTP/HTTP2.html.
4. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach et T. Berners-Lee, "RFC2616: HyperText Transfer Protocol—HTTP/1.1", Network Working Group (1999).
5. Références provenant de différentes sources, citées sur la page http://usability.gov/guidelines/softhard.html : Anna Bouch, Allan Kuchinsky et Nina Bhatti, "Quality Is in the Eye of the Beholder: Meeting Users' Requirements for Internet Quality of Service", CHI (2000) ; Martin Corl, "System Response Time Effects on User Productivity", *Behaviour and Information Technology*, vol. 5 n° 1, pages 3-13 (1986) ; Jakob Nielsen, "Top Ten Mistakes in Web Design" (1996), http://www.useit.com/alertbox/9605.html ; Nielsen, "The Need for Speed" (1997), http://www.useit.com/alertbox/ 9703a.html ; Nielsen, "Changes in Web Usability Since 1994" (1997),

http://www.useit.com/alertbox/9712a.html ; Nielsen, "The Top Ten New Mistakes of Web Design" (1999), http:// www.useit.com/alertbox/990530.html.

6. Kristol et Montulli, "RFC2109: HTTP State Management Mechanism", Network Working Group (1997).
7. Martin Pool, "Privacy Problems with HTTP Cache-Control", Bugtraq (2000). http:// cert.uni-stuttgart.de/archive/bugtraq/2000/03/msg00365.html.
8. Bamshad Mobasher, Robert Cooley et Jaideep Srivastava, "Automatic Personalization Based on Web Usage Mining", ACM Communications, vol. 43 n° 8, p. 142-151 (1999).
9. Edward Felten et Michael Schneider, "Timing Attacks on Web Privacy", ACM Conference on Computing and Communications Security (2000).

Chapitre 15

1. ISO/IEC Standard 9899, "Programming Language – C" (1999). http://plg.uwaterloo.ca/~cforall/N843.ps.
2. DISCO. http://www.altmode.com/disco.

Chapitre 16

1. Albert Laszlo Barabasz, Vincent W. Freeh, Hawoong Jeong et Jay B. Brochman, "Parasitic Computing", Lettre au journal *Nature*, n° 412 (2001).
2. Leech, "RFC 3607: Chinese Lottery Cryptoanalysis Revisited", Network Working Group (2003).

Chapitre 17

1. Bill Cheswick, Hal Burch et Steve Branigan, "Mapping and Visualizing the Internet" (2000). http://www.cheswick.com/ches/papers/mapping.ps.gz.
2. Despoof : http://razor.bindview.com/tools/desc/despoof_readme.html.
3. Hal Brunch et Bill Cheswick, "Tracing Anonymous Packets to Their Approximate Source", (2000). http://www.usenix.org/publications/library/ proceedings/lisa2000/ burch/burch_html.

Chapitre 18

1. Lance Spitzner, *Honeypots: Tracking Hackers*, Addison-Wesley (2002).
2. Jose Nazario, *Defense and Detection Strategies against Internet Worms*, Artech House (2003).
3. Michal Zalewski, "Museum of Broken Packets", (2001). http:// lcamtuf.coredump.cx/mobp.

Index

A

Accumulateurs 56
ACK
  paquets et flags
    attaque DoS 304
    despoofing 304
    en TCP 155
    et pare-feu sans états 212, 219
    idle scan 232
  valeurs dans les mémoires tampon 229
Address Resolution Protocol (ARP) 119
Adleman, Len 9
Adresse matérielle (MAC) 67, 110
Algorithmes
  GCL 12
  GNFS 10
  PRNG 11
  RSA 10
  Viterbi 22
Analyse comportementale 240
  des attaques 266
  des navigateurs Web 259
Analyse de l'effort de calcul 61
AND 30
Anderson, Josh 113
Applets Java 276
Arkin, Ofir 113, 164
ARP (Address Resolution Protocol) 119
Attaques
  DoS
    et PMTUD 222
    identifier l'origine 294
    paquets et flags ACK 304
    retombées 305
  mesures de l'attaquant 266
  par recoupement de fragmentation 209
Attracteurs des numéros de séquence 202

B

Barabasz, Albert Laszlo 272
Berners-Lee, Tim 241
BGP (Boundary Gateway Protocol) 140
Biham, E. 62
Blind spoofing 178
Blinding 61
Blinkenlights 96
  définition 95
  espionnage des DEL via LPT 99
  espionner 103
  implications 96
  protection 106
Boole, George 30
Booléens 34
  De Morgan 32
  et flip-flop 42
Boundary Gateway Protocol (BGP) 140
Brochman, Jay B 272
Brouillage du signal 92
Brunch, Hal 295
Bush, Vannevar 241

C

CAIDA, carte d'Internet 290
Calcul
  parasitaire 275
  volatil 287
CAM (Content Addressable Memory) 119
  débordement 123
Camoufler l'identité d'un logiciel 240
Capteurs CCD (Charge Coupled Device) 26
Carrier Sense Multiple Access with Collision Detection (CSMA/CD) 91
Cartographie d'Internet 292
CERN 241
Checksum 91
Cheswick, Bill 290, 295
Church et Turing (thèse) 43
Codage
  bipolaire 91
  Manchester (ou biphase) 81
  NRZ 82
Collisions
  dans les transmissions de données 92
  numéros de séquence 187

Conception flip-flop 42
Conception multicycles 47
Conseil européen pour la recherche nucléaire (CERN) 241
Contrôleur programmable d'interruption (PIC) 12
Cooke, Jean-Luc 276
Cookies
HTTP 250
SYN 211
Coordonnées temporisées 190
Courtay, Olivier 171
Crible général de corps de nombres (GNFS) 10
CRONOS 171
Cryptographie
à clé publique 10
et problèmes NP 273
RSA 10
CSMA/CD (Carrier Sense Multiple Access with Collision Detection) 91

D

De Morgan, Augustus 30
Défauts de cache 51
défauts de cache 51
DEL 96
espionner 103
implications 96
protection 106
Despoof 294
Détournement de connexion 178
DF (don't fragment) 148
controverse 222
et fingerprinting passif 165
Dibits 86
Diffie, Whitfield 9
Diodes électroluminescentes 95
DISCO 267
Discriminateur binaire 103
DNS, problèmes 204
DoS 220
DPSK (modulation par sauts de phase) 84
DRAM (mémoire vive dynamique à accès direct) 50
DSL (Digital Subscriber Line) 82
DTP (Dynamic Trunking Protocol) 120, 124

E

early-out 56
En-têtes
HTTP 243
ICMP 162
IP 151
numéro d'identification 149
offset 148
TCP 162
MSS (Maximum Segment Size) 160
UDP 153
EOL 161
Espaces d'adressage
et PMTUD 222
Ethernet 110
Internet 142
Étapes d'exécution
dans le temps 56
des instructions 49
Ethernet 111
composants 95
mémoire tampon 114
remplissage des trames 112, 114
réseaux commutés 117
somme de contrôle 92
Euler, théorème 10

F

Factorisation GNFS 10
Failles du wi-fi 134
Felten, Edward 257
Fielding, Roy T 242
Fingerprinting passif
applications 176
optimisation du contenu 175
avantages et implications 186
empêcher 177, 204
en pratique 174
flag DF 165
MSS 169
p0f 175
techniques 171
Flags
dans les en-têtes IP 148
dans les en-têtes TCP 158
DF (don't fragment) 148
controverse 222
et fingerprinting passif 165
MF (more fragments) 148
URG 159
Flip-flop 42
Fonctions de hachage 17
Fragmentation
attaque par recoupement 209
dans les filtres sans états 209
paquets IP 148, 182
Freeh, Vincent W 272
Frystyk, Henrik 242
FSK (Frequency Shift Keying) 83

FTP (File Transfer Protocol) 215
Fyodor 164

G

Générateur de congruence linéaire (GCL) 12
Génération de nombres aléatoires
automatique 11
et scan des ports 266
matérielle 26
GNFS 10
Goulets d'étranglement HTTP 245
GUID (Globally Unique Identifier) 67

H

Hachage 17
Heen, Olivier 171
Hellman, Martin 9
Hijacking 178
Horloges
et scan des ports 265
et transmision des données 80
HTML (Hypertext Markup Language) 241
HTTP (Hypertext Transfer Protocol) 241, 244
cache 248
confidentialité 259
cookies 250
en-têtes 243, 244
goulets d'étranglement 245
latence 246
poignée de main 245
recherche séquentielle 245

I

ICMP 163
échec du PMTUD 220
en-têtes 162
stockage parasitaire 278
Idle scan 235
IMUL 56
Interface LPT, espionnage des DEL 99
Internet
cartographie 292
Control Message (ICMP) 163
Explorer 242
Protocol (IP) 151
Interruptions lmatérielles IRQ 12
IP 151
fragmentation 148, 182
IP ID 149
utilisation pour le profiling 236
IP spoofing 146
détournement de la connexion 178
IPv4 141
IRQ (Interrupt Request) 12
ISNProber (utilitaire) 203

J

Jeong, Hawoong 272
Jun, Benjamin 25

K

Kaminsky, Dan 205
Khan, Saqib A 278
Kocher, Paul 25
Krawczyk, H 263

L

Logique booléenne 30
calculs 40
équations SAT 275
et conception informatique 37
et flip-flop 42
mise en application 34
Loi de De Morgan 32
Loveless, Mark 292
Lughry, Joe 96

M

MAC (Media Access Control) 110
et protocole Ethernet 118
spoofing 123
Machine de Turing 45
Markov, modèle caché 21
Masquerading 216
McLachlan, Donald 292
MDF (modulation par déplacement de fréquence) 83
Mécanisme /dev/random 19
Memex 241
Mémoire
acoustique 277
cache 51
tampon
fuites 69
pour les trames Ethernet 114
stockage parasitaire 283
valeurs ACK et URG 229
Métadonnées stockées dans les documents 68
MF (more fragments) 148
Microsoft Word, stockage des métadonnées 67

Modèle
de Markov caché 21
OSI 112
Modems 82, 90
Modulation
de l'impulsion 103
par déplacement de fréquence (MDF) 83
par sauts de phase (DPSK) 84
Modulo 12, 55
Morris, Robert T 178
Mosaic 242
Mozilla 242
MSS (Maximum Segment Size)
dans les en-têtes TCP 160
et fingerprinting passif 169
et pare-feu 218
MSS clamping 217
MTU (maximum transmission unit) 145
et pare-feu 217
fingerprinting passif 169

N

NAND 32
NAT 215
Navigateurs Web
analyse comportementale 259
historique 242
Nehemiah 25
Netscape Navigator 242
NMAP 164, 235
Nombres aléatoires
génération
automatique 11
matérielle 26
Nombres premiers en cryptographie 10
NOP 161
NOR 32
NOT 30
NP 275
NRZ 82
NUL (scan) 211
Numéro d'identification (ID)
en-têtes IP 149
et idle scan 234
utilisation pour le profiling 236
Numéros d'acquittement TCP 158
Numéros de séquence
attracteurs 202
TCP 158

O

Observations
des attaquants 267
directes 303
Offset (paramètre) d'en-tête IP 148
Offset des données TCP 158
Olsson, Mikael 214
Opera 169
Opérateurs booléens 34
loi de De Morgan 32
Optimisation
early-out 59
fingerprinting passif 175
Options TCP 162
OR 30
conception flip-flop 42
loi de De Morgan 30
OSI 112

P

p0f 175
Paramètres d'en-tête TCP 159
Pare-feu 208, 212
à états 212
et réponses inattendues 219
et masquerading 216
filtrage sans états 212
MTU 217
réécriture des paquets 215
taille des segments 218
traduction d'adresse 215
Path MTU Discovery (PMTUD) 219
Phototransistor 98
PIC 12
ping 162
de la mort 220
Pipelining 53
PMTU 148
exemples d'échec 222
Poignée de main
HTTP 245
TCP 154
Pool, Martin 250
Ports éphémères 152, 216
Postel, Jon 178
Pot de miel 175
Prédiction de branchement 53
PRNG
matériels 26
scan des ports 266
sécurité 11, 15
Projet CRONOS 171
Projet Skitter 290
Protocoles
BGP 140
Ethernet 111

composants 95
remplissage des trames 114
réseaux commutés 117
FTP 215
HTTP 244
cache 248
cookies 250
latence 246
IP 151
fragmentation 148, 182
TCP 162
UDP 153

PSYLock 130

Purczynski, Wojciech 279

Q

QAM (modulation d'amplitude en quadrature) 87

R

Raymond, Eric S 96

Rayonnements électromagnétiques 66

Reed, Tracy 133

Remplissage des trames Ethernet 112

Réseaux
commutés et résolution d'adresse 120
triangulation 294
wi-fi 134

Rivest, Ron 9

Roualland, Gaël 177

Routeur backbone 140

RSA 9, 10
décryptage 56

S

Saffroy, Jean-Marc 177

Scan des ports 233
analyse 266
idle scan 235
NMAP 164, 235
NUL et Xmas 211

Schneider, Michael 257

SGML (Standard Generalized Markup Language) 241

Shamir, Adi 9, 62

Signal, mécanisme de brouillage 92

Somme de contrôle
des fragments IP 182
en-têtes
ICMP 163
IP 150
TCP 159
et équations SAT 274
Ethernet 92

Spitzner, Lance 175

Spoofing en aveugle 155

Staging 48
dans le temps 56
des instructions 49

Stewart, Joe 205

Stockage parasitaire 278, 288
files d'attentes 286
mémoire tampon 283

SYN, cookies 211

Système
autonome 140
binaire 34

Szymanski, Jacek P 222

T

TCP 162
flags 158
offset des données 158
options 162
EOL 161
NOP 161
paramètres d'en-tête 159
poignée de main 154

TEMPEST 66

Théorèmes
d'Euler 10
des restes chinois 10

Thèse de Church et Turing 43

Traceroute 147

Traduction d'adresse 215

Transmission Control Protocol (TCP) 162

Triangulation 294

Ts'o, Theodore 19

TTL 146

Turing (machine) 45
Turing tarpits 46
UTM 45

Turing, Alan 8, 43

U

UAL 49

UDP 153

Umphress, David A. 96

URG (flag) 159

URI (Universal Resource Identifier) 242

USB (Universal Serial Bus) 90

User Datagramm Protocol (UDP) 153

Usurpation d'identité 178

UTM (Machine de Turing universelle) 45

V

Van Eck, Wim 64

Veysset, Franck 171

Viterbi 22

W

W3C (World Wide Web Consortium) 241

Warflying et Wardriving 134

WEP (Wired Equivalent Privacy) 132

Wi-fi, failles 134

Wood, Preston 267

World Wide Web Consortium (W3C) 241

X

Xmas (scan) 211

XOR 38

Menaces sur le réseau

Sécurité informatique : guide pratique des attaques passives et indirectes

Des récits révélateurs qui éclairent d'un jour nouveau les failles et les risques de l'informatique au quotidien

Dans cet ouvrage unique, Michal Zalewski nous plonge dans les entrailles de l'informatique moderne et porte un jour nouveau sur la conception d'un réseau et nos propres pratiques informatiques. À travers l'étude de quelques défis exceptionnels, rares et souvent très recherchés en terme de sécurité, ce récit fascinant échappe à toute classification et renonce à l'opposition traditionnelle entre attaque et victime.

Michal Zalewski est depuis longtemps connu et respecté dans les communautés de hackers et de sécurité pour son intelligence, sa curiosité et sa créativité. Dans *Menaces sur le réseau*, il partage son savoir-faire et son expérience pour expliquer le fonctionnement des ordinateurs et des réseaux, le traitement et la livraison des informations ainsi que les menaces pour la sécurité qui se cachent dans l'ombre. Là où d'autres livres se contentent d'énumérer les failles de sécurité, Zalewski les explique.

La lecture de cet ouvrage fascinera les étudiants et les professionnels de la sécurité, les responsables SI et tous les technophiles désireux d'apprendre comment la sécurité informatique s'inscrit dans un cadre plus large.

À propos de l'auteur

Michal Zalewski, chercheur autodidacte sur la sécurité de l'information, a travaillé sur des sujets allant de la conception de matériel et de systèmes d'exploitation à la gestion des réseaux. Depuis le milieu des années 1990, il participe activement à Bugtraq, une liste de diffusion dédiée à la sécurité informatique. Il a travaillé comme expert pour la sécurité dans plusieurs entreprises de renom, dont deux géants des télécommunications, à la fois dans son pays natal la Pologne et aux États-Unis.

Niveau : Intermédiaire / Avancé

Catégorie : Sécurité informatique

PEARSON

Pearson Education France
47 bis, rue des Vinaigriers 75010 Paris
Tél. : 01 72 74 90 00
Fax : 01 42 05 22 17
www.pearson.fr

NO STARCH PRESS

ISBN : 978-2-7440-4031-3

9 782744 040313